Physics Informed Neural Networks
Applied for solving differential equations

Thibaud Martin, Arthur Lamour and Justin Manson

Abstract

In our ML4Science project for the Machine Learning class (CS-433), we explore the use of Physics-Informed Neural Networks (PINN) to solve differential equations in physics and engineering. PINNs are a type of neural network that incorporate physical constraints and laws into their architecture, allowing them to solve complex differential equations with high accuracy. We apply PINNs to three differential equations: the Cauchy problem, the viscous Burgers' equation, and the heat equation. In each case, we describe the specific methods and approaches we used to train and evaluate the PINN models, and we discuss the results we obtained. Our results demonstrate the potential of the PINN as a powerful tool for solving differential equations, while also outlining its limitations.

1 Introduction

Differential equations are a fundamental tool for describing the behavior of physical systems, and they play a key role in many areas of science and engineering. However, solving differential equations can be challenging, particularly when the equations are complex or the data is limited. Traditional methods such as analytical techniques or numerical methods may not always be feasible or accurate.

In this context, Physics-Informed Neural Networks (PINNs) offer a promising alternative. By incorporating physical knowledge into the training process, PINNs can provide accurate solutions to differential equations using little data, or even none in certain cases.

In this work we leverage the use of PINNs for solving two tasks.

1. Solution inference: model predicts the solution of a given differential equation system for any point in its domain.
2. Problem identification: model predicts the values of unknown parameters within the differential equation system given its solution.

Using [6] as a starting point, we initially reproduced work done with Burgers equation. We developed our own implementation of a problem inference and identification, results of which are outlined in section 3. The bulk of our work however, is an exploration of the strengths and limitations of the PINN model for solution inference. Our axes of exploration include the dimensional space of problem parameters, the frequency of the desired solution, the ability to predict solutions outside the training domain and the effect of model architecture.

To this aim, our work is split in three sub-sections. The first focuses on the one-dimensional Cauchy problem, the second is dedicated to the two-dimensional burgers problem and heat equation, and the third adds a layer of complexity by converting a constant parameter from the standard heat equation to a variable so as to generate a three-dimensional problem.

2 One dimensional Cauchy problem

2.1 Equation structure and solution

Initially, our aim was to show how to implement a physics informed neural network and affirm its performance over a simple problem. One dimension is easiest to visualize, and less complex for a model to train, so we started by implementing a PINN for an ordinary differential equation, namely a Cauchy problem [2] as shown below.

$$\begin{cases} y''(t) + 2y'(t) + y(t) = h(t) & t > 0 \\ y(0) = 1,\ y'(0) = -1 \end{cases}$$

We treated the particular case where $h(t) = \frac{1}{4}\alpha^2 \cdot e^{-t} \cdot \cos(\alpha t)$. Where α is a constant we can tune to adjust the frequency of the desired solution. The corresponding solution for the Cauchy problem is found by using the Laplace transform and its properties. The result is shown below:

$$y_{exact}(t) = \frac{1}{4}e^{-t} \cdot (5 - \cos(\alpha t))$$

Aware that the PINN would converge to a solution more easily for a one dimensional problem (given its reduced complexity), we wanted to test the limits of the models capabilities by increasing the frequency of the desired solution. We therefore trained three models with the distinct values of $\alpha \in [1, 10, 20]$.

The expected result in each case is an exponential decay with small oscillations of increasing frequency as α increases.

2.2 Model architecture and loss function

Our aim is to train a model for which we can input a time t and obtain a prediction for the solution at that point $y(t)$. The neural network is therefore composed of input and output layers containing a single neuron each. In between are 7 hidden layers with 20 neurons each and a hyperbolic tangent activation function.

Using the main equation from the Cauchy problem, we define:

$$f(t) = y''(t) + 2y'(t) + y(t) - \frac{1}{4}\alpha^2 \cdot e^{-t} \cdot \cos(\alpha t)$$

We say that $f(t)$ encodes the physics of the Cauchy problem. We can train the model by evaluating $f(t)$ for an array of collocation points and penalizing predictions that differ from the trivial solution. The loss itself is defined as a sum of mean squared errors over collocation points as well as the mean squared error for each initial condition: $Loss = MSE_f + MSE_{bc} + MSE_{ic}$, and the individual terms are defined as:

$$\begin{cases} MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i)|^2 \\ MSE_{ic} = |y(0) - 1|^2 + |y'(0) + 1|^2 \end{cases}$$

2.3 Results

We restricted the time domain to $t \in [0, 8]$ to study only the region where the function varies. We trained each of the three models using $N_f = 10,000$ collocation points sampled randomly throughout the domain of definition, and 4000 iterations with the Adams optimizer. The result is shown in figure 1.

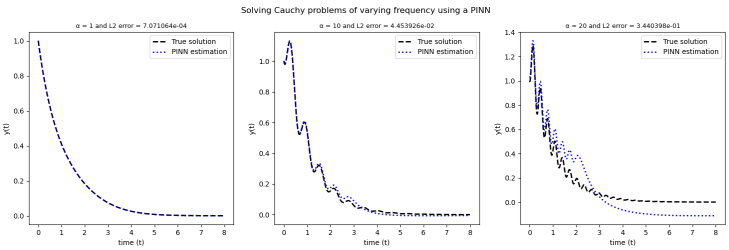


Figure 1: Plot of predicted and exact solution for Cauchy problems of varying frequency ($\alpha = 1, 10, 20$ from left to right).

Figure 1 shows that the PINN performs near impeccably for the trivial case where $\alpha = 1$, generating a prediction with an L_2 error of $7.0711 \cdot 10^{-4}$. The performance diminishes somewhat for the case where $\alpha = 10$, yielding an L_2 error of $4.4539 \cdot 10^{-2}$. Finally model fails to converge to the true solution altogether in the case where $\alpha = 20$, yielding an L_2 error in the 10^{-1} order of magnitude which is insufficient for most applications.

These results show that the PINN delivers a robust prediction in one dimension, but fails when the frequency of the desired solution is increased drastically. We'd argue, however, that the PINN has limited utility for solving such ordinary differential equations. There exist other methods such as the Runge Kunta method [1] that deliver better or faster solution approximations. The interest of the PINN lies in its application to two dimensional partial differential equations (PDEs), where it is able to deliver fairly accurate predictions faster than current solvers. This will be discussed in the following sections.

2.4 Inference extrapolation on a modified Cauchy problem

The aim in this section is to train an inference model for a one-dimensional Cauchy problem with collocation points restricted to a certain domain, and extrapolate by testing the models ability to accurately predict the solution for points that lie outside the training domain.

To this aim, we define the following modified Cauchy problem:

$$\begin{cases} 3y''(t) + 2y'(t) + y(t) = -2\sqrt{2}\sin(t + \frac{\pi}{4}) & t > 0 \\ y(0) = 1,\ y'(0) = 0 \end{cases}$$

The corresponding exact solution is $y_{exact}(t) = \cos(t)$.

The inference model architecture largely remains the same as the one used in the previous sections, however the collocation points used for training are restricted to the following domain: $\{t_f^i\}_{i=1}^{N_f} \in [0, 12\pi]$.

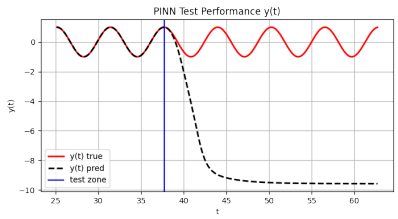


Figure 2: Prediction of $y(t)$ - trained for $t \in [0, 12\pi]$ and extrapolated to $t \in [8\pi, 20\pi]$.

The vertical blue line in figure 2 highlights the upper bound for collocation points used to train the model. The figure shows that the model quickly fails to provide a prediction outside of its training domain, which shows that the PINN in its basic form cannot be used for extrapolation. This behavior can be improved by using Dynamic Pulling Method [3].

3 Two dimensional Burgers' equation

While the Cauchy problem can be solved using analytical techniques such as the Fourier or Laplace transforms, the viscous Burgers' equation presents a much greater challenge, as it is a nonlinear partial differential equation. Analytical methods are generally not effective for solving this type of equation. Numerical methods such as finite difference methods and finite element methods can be used to solve the viscous Burgers' equation, but they have some limitations. These methods involve discretizing the domain of the equation into a grid of points and approximating the solution at each point using a mathematical formula. While they can provide accurate solutions in some cases, they may require a large amount of data and computation, which can be expensive and time-consuming. In contrast, PINNs can rely on small amount of data and learn to approximate the solution, which makes them more accurate and efficient.

In the following subsections, we aimed to reproduce the inference performed in [5] and identification in [4] using PyTorch instead of Tensorflow, more details can be found in section 6.2.

3.1 Inference

We chose N_u randomly selected points for the initial and boundary conditions, and N_f collocations points generated using *Latin Hypercube Sampling* strategy, which aims to evenly distribute the points within the domain, while also minimizing the correlation.
The model uses hyperbolic tangent as activation function. The weights are initialized using Xavier initialization to prevent vanishing or exploding gradients, and improve the convergence rate. We chose LBFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) over Adam, because it requires fewer function evaluations and gradient evaluations per iteration, making it converges faster and gave more accurate results. Also LBFGS could be prefered for large-scale optimization problems such as solving nonlinear PDE with a high number of collocations points. We performed two systematic studies. First, we varied the network architecture with $N_u = 100$ and $N_f = 10000$ and compared the models predictions to the exact solution. As expected, the smallest errors were obtained on the model with highest explicative power, the highest number of layers and neurons. The results can be found in the following table 1. The second study, we varied N_u and N_f, with a PINN of 9 hidden layers and 20 neurons. Similarly, the model performed better with more data, the results can be found in the following table 2.

Layers \ Neurons	10	20	40
2	8.7e-03	1.4e-03	3.5e-03
4	9.5e-05	2.6e-04	3.0e-05
6	9.7e-06	1.9e-05	6.3e-07
8	1.3e-06	4.0e-05	1.7e-06
10	1.6e-06	5.5e-06	3.9e-07

Table 1: MSE_u between the exact and predicted solution for different architectures

N_u \ N_f	1000	2000	5000	10000
10	9.81e-01	2.8e-01	5.1e-01	6.9e-02
50	7.1e-02	4.0e-02	1.1e-04	2.4e-05
100	5.2e-02	9.5e-06	1.3e-04	5.6e-07
200	3.0e-02	7.1e-03	4.3e-07	6.1e-06

Table 2: MSE_u between the exact and predicted solution for different training data sizes

3.2 Identification

The viscous Burgers' equation can be written as $\frac{\partial u(t,x)}{\partial t} + \lambda_1 u(t,x)\frac{\partial u(t,x)}{\partial x} - \lambda_2 \frac{\partial^2 u(t,x)}{\partial x^2} = 0$ where λ_1 and λ_2 are unknown real parameters to estimate. In parameters identification, it is assumed that the solution $u(t,x)$ is known, and we want to find the PDE's parameters.

We performed two systemic studies. First, we keep $N = 2000$ the number of training data points, and we vary the network architecture and look for the smallest relative errors on λ_1 and λ_2. From table 3, we observe that models with high explicative power have the smallest relative errors. Secondly, we add different levels of uncorrelated noise to the training data. We can see that up to 10% the relative error with respect to λ_1 remains small, less than 2%, but for high noise level, the error becomes very high on λ_2, as it mutlitplies the second derivative term, making it very sensible to noise. 4.

	% error in λ_1			% error in λ_2		
Layers \ Neurons	10	20	40	10	20	40
2	11.3	2.34	1.92	128	28.3	14.8
4	0.18	0.06	0.11	1.92	2.08	1.46
6	0.07	0.001	0.05	0.64	0.70	0.86
8	0.07	0.03	0.03	0.57	0.15	0.81
10	0.33	0.06	0.01	1.38	0.04	0.80

Table 3: Relative error on the parameters based on the network architecture

	% error in λ_1				% error in λ_2			
Noise / N_u	0%	1%	5%	10%	0%	1%	5%	10%
500	0.02	0.10	0.84	3.51	0.26	0.50	31.0	34.7
1000	0.02	0.05	3.04	0.002	2.05	0.79	44.1	9.10
1500	0.13	0.03	0.28	0.72	0.46	0.95	1.36	2.11
2000	0.04	0.02	0.90	1.81	0.31	1.79	0.35	0.21

Table 4: Relative error on the parameters with different training size and added noise

4 Two dimensional Heat equation

4.1 Equation structure and general solution

Let $a \neq 0$ and $l > 0$ be constants and $h : [0, L] \to \mathbb{R}$ be a C^1 function, such that $h(0) = h(L) = 0$. The Heat equation [2] is written as:

$$\begin{cases} \frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2} & x \in (0, L),\ t > 0 \\ u(0, t) = u(L, t) = 0 & t > 0 \\ u(x, 0) = h(x) & x \in (0, L) \end{cases}$$

Its' general solution is given by:

$$u(x,t) = \sum_{n=1}^{\infty} \alpha_n \sin\left(\frac{n\pi}{L}x\right) e^{-a^2\left(\frac{n\pi}{L}\right)^2 t},\ \alpha_n = \frac{2}{L}\int_0^L h(y)\sin\left(\frac{n\pi}{L}y\right)\,dy$$

If the initial condition $h(x)$ is given under the form of a Fourier Series, then the coefficients α_n are determined by comparing the expressions $u(x, 0)$ and $h(x)$ term by term.

4.2 PINN inference for a specific solution of the heat equation

The heat equation describes a smooth diffusion process and is therefore, in general, an easier problem for training a PINN when compared to the Burgers equation. In order to add some complexity to our model, we chose a particular solution containing a linear combination of two sinusoids of differing frequency.

The chosen initial condition is $h(x) = 2\sin\left(\frac{\alpha\pi}{L}x\right) - \sin\left(\frac{3\alpha\pi}{L}x\right)$ while the constants are $a = \frac{1}{2}$ and $L = 2$. The α is a parameter we can tune to modify the frequency of the desired solution. The solution of the Heat equation for this particular case is given by:

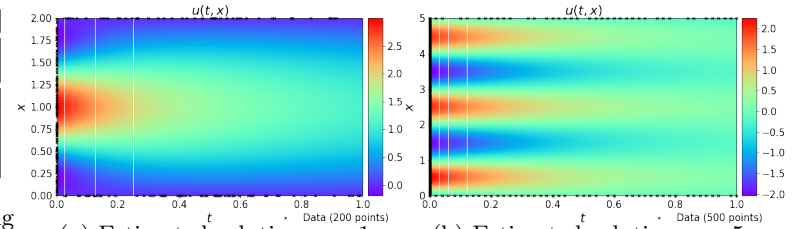
$$u_{exact}(x,t) = 2\sin\left(\frac{\alpha\pi}{2}x\right) e^{-\frac{\alpha^2\pi^2}{16}t} - \sin\left(\frac{3\alpha\pi}{2}x\right) e^{-\frac{9\alpha^2\pi^2}{16}t}$$

The term $f(x,t) = \frac{\partial u}{\partial t} - \frac{1}{4}\frac{\partial^2 u}{\partial x^2}$ encodes the physics of the heat equation. The solution $u(x,t)$ is learned by minimizing the mean squared error loss: $Loss = MSE_f + MSE_{bc} + MSE_{ic}$. The individual loss terms are given as:

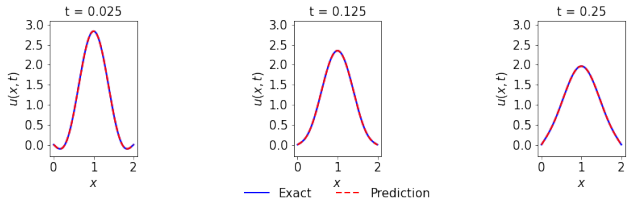
$$\begin{cases} MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(x_f^i, t_f^i)|^2 \\ MSE_{bc} = \frac{1}{N_{bc}}\sum_{i=1}^{N_{bc}} |u(0, t_{bc}^i)|^2 + \frac{1}{N_{bc}}\sum_{i=1}^{N_{bc}} |u(2, t_{bc}^i)|^2 \\ MSE_{ic} = \frac{1}{N_{ic}}\sum_{i=1}^{N_{ic}} |u(x_{ic}^i, 0) - h(x_{ic}^i)|^2 \end{cases}$$

Where N_f, N_{bc}, N_{ic} are the number of collocation points, boundary condition and initial condition points respectively evaluated to obtain the loss.

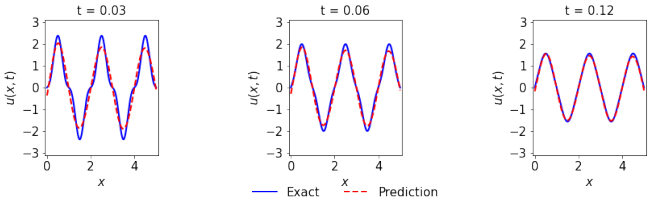
The model architecture was composed of 9 hidden layers with 20 neurons each, all using a hyperbolic tangent activation function. The input layer consisted of two neurons (for variables x and t), while the output layer consisted of a single prediction for $u(x,t)$. The loss was evaluated using $N_f = 10,000$ collocation points, $N_{bc} = 100$ points for each of the two boundary conditions (Dirichlet conditions) and $N_{ic} = 100$ points for the initial condition. Training was done using the LBFGS optimizer over 50,000 iterations or until the loss dropped below a 10^{-5} threshold. We built models for two solutions of differing frequencies, those are $(\alpha = 1, L = 2)$ and $(\alpha = 5, L = 5)$, the second model was trained with $N_{ic} = 500$ and $N_{bc} = 50$, to make it more precise at $t = 0$ as the initial condition is more complex, but less precise on the boundaries due to the unbalance between initial and boundary training data. The results are shown in figure 3.



(a) Estimated solution $\alpha = 1$ (b) Estimated solution $\alpha = 5$



(c) Cutouts from figure 3a.



(d) Cutouts from figure 3b.

Figure 3: Heat equation results.

The low frequency model ($\alpha = 1$) obtains an L_2 error of $5.3668 \cdot 10^{-4}$ when tested with the points used to generate figure 3a. This statistic shows that the PINN performs well for two dimensional problems. In fact, figure 3a provides a realistic heat diffusion gradient, while the cutout at $t = 0.025$ in figure 3c confirms that the quality of the prediction remains near impeccable when the function oscillates more, with the two lines remaining superposed throughout the domain of $x \in [0, 2]$.

The high frequency model ($\alpha = 5$) obtains an L_2 error of $1.9359 \cdot 10^{-1}$ when tested with the points used to generate figure 3b. As with the one-dimensional case, we notice that the PINN yields less accurate results when attempting to predict a high frequency solution. This model particularly struggled to accurately predict points near the initial condition as can be seen on the cutout at $t = 0.03$ in figure 3d. This is understandable as the source of a diffusion is the zone with the steepest gradient, and thus the most complex for our model to learn. To mitigate errors near the initial condition, we added collocation points in this region (and thus reduced the number of points at the boundary condition). This is reflected in the cutouts in figure 3d, as the prediction strugles to match the exact solution at the boundary conditions.

5 Heat equation augmented to three dimensions

In order to develop a three dimensional PDE for which we could easily generate an exact solution (to evaluate our model), we started from the Heat equation defined in section 4.1 and redefined the constant $a \neq 0$ as a variable, such that $a \in [0, 1]$. The Heat equation becomes:

$$\begin{cases} \frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2} & x \in (0, L),\ t > 0,\ a \in [0, 1] \\ u(0, t, a) = u(L, t, a) = 0 & t > 0,\ a \in [0, 1] \\ u(x, 0, a) = h(x) & x \in (0, L),\ a \in [0, 1] \end{cases}$$

We evaluated the particular case where $h(x) = 2\sin(\alpha\frac{\pi}{L}x)$ and $L = \pi$. Note that α is a constant we can tune to adjust the frequency of the desired solution. The solution of the three dimensional Heat equation for this particular case is given by:

$$u_{exact}(x, t, a) = 2\sin(\alpha \cdot x) \cdot e^{-a^2 t}$$

In the three dimensional case, the physics of the heat equation is encoded in the term $f(x, t, a) = \frac{\partial u}{\partial t} - a^2\frac{\partial^2 u}{\partial x^2}$. The solution $u(x, t, a)$ is learned by minimizing the modified loss:

$$Loss = MSE_f + MSE_{bc} + MSE_{ic} + MSE_{sup}$$

The individual loss terms are given as:

$$\begin{cases} MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(x_f^i, t_f^i, a_f^i)|^2 \\ MSE_{bc} = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} |u(0, t_{bc}^i, a_{bc}^i)|^2 + \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} |u(\pi, t_{bc}^i, a_{bc}^i)|^2 \\ MSE_{ic} = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |u(x_{ic}^i, 0, a_{bc}^i) - h(x_{ic}^i)|^2 \\ MSE_{sup} = \frac{1}{N_{sup}} \sum_{i=1}^{N_{sup}} |u(x_{sup}^i, t_{sup}^i, a_{sup}^i) - u_{exact}(x_{sup}^i, t_{sup}^i, a_{sup}^i)|^2 \end{cases}$$

Previous work in [6] had stated that PINNs work well when solving two dimensional problems but generalize badly to three dimensions. The MSE_{sup} term is therefore added to assist the PINN in converging to a solution. This term contains the mean squared error of N_{sup} points, randomly predicted over the domain of definition, with respect to the known exact solution $u_{exact}(x_{sup}^i, t_{sup}^i, a_{sup}^i)$. The aim is to experiment with the number of support points N_{sup} provided to the model to evaluate their effect on model accuracy.

The model was trained using $N_f = 40^3$ collocation points, $N_{bc} = 100^2$ points for each of the two boundary conditions and $N_{ic} = 100^2$ points for the initial condition. All the points were generated randomly in their interval. Regarding the model architecture, we used 7 hidden layers with 20 neurons each, the model was trained with LBFGS optimizer with the same parameter as in 4.2. Models were compared to the true solution on a cube with 100 side length.

We experimented with 2 different values of α (solution frequency variation) and displayed cross-sections of the prediction for all permutations of three distinct instances of variables t and a.

$$t, a \in [0.25, 0.5, 0.75] \ ; \ \alpha \in [1, 2]$$

Using this new formula for the loss and the t, a values described above, we fitted the model with varying numbers of support points in [0, 10, 20, 40]. The plots for the $\alpha = 1$ can be found on the Appendix. From those plots we observed that with a lower frequency, the model obtained a really good accuracy, almost perfectly matching the true function on all points, even with no support points. However, in the case of $\alpha = 2$, as the solution oscillates more, the model cannot find the true function as easily. The figure below shows the results obtained with no support points.

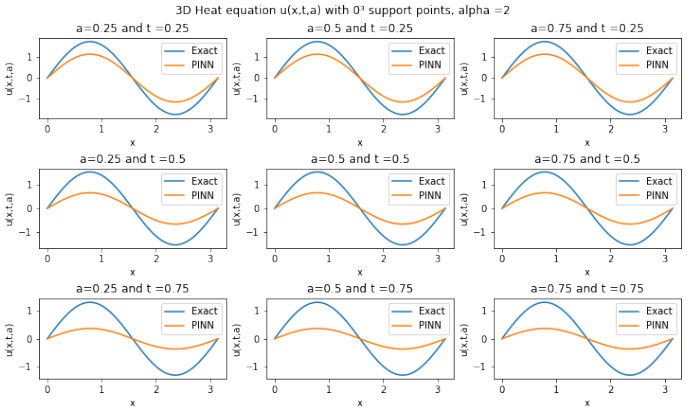


Figure 4: Comparison of predicted and exact $u(x, t, a)$ with no support points

As we can see on this figure, the model obtains a relatively good prediction for points that are close to the boundary conditions (a=0, t=0), but fails when a and/or t grows bigger. The model receives a lot of data on the initial and boundary conditions, which helps it to quickly find a good approximation when a and t are small. However, it then fails to optimize for the larger value, because the loss is stuck in this local minima. This is where including support points becomes useful. The next figure was obtained using 40^3 support points.

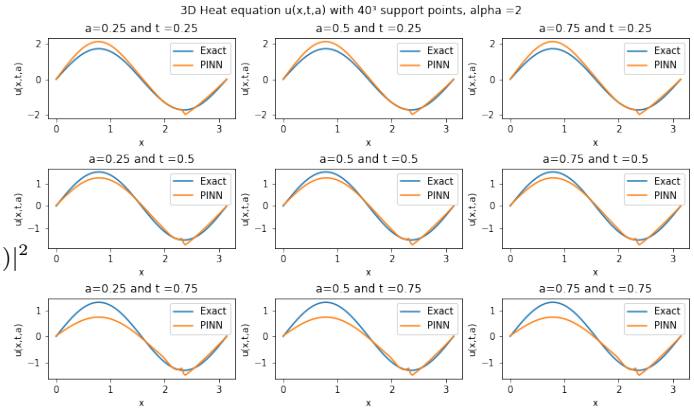


Figure 5: Comparison of predicted and exact $u(x, t, a)$ with 40^3 support points

We can easily observe from those 2 figures how adding support points helps a lot the model to converge to a better solution, in particular as we get further from the boundary conditions. We noticed however that, even with the support points, the solution is far from perfect and some artifacts can be observed. We also noticed during the training that the MSU_{sup} term is one order of magnitude higher than the two others, which indicates that it is limiting factor in the training process. Additional figures with 10^3 and 40^3 support points are present on the appendix. Unfortunately, as the support points are generated using the true solution, one couldn't use this approach in an experiment where it is unknown, and we observed that the performance is relatively poor without them. An alternative would be, if empirical data is available, to use it in place of support points. It might be interesting in future studies to specifically select support points in the areas far from the boundary conditions, as this is where the PINN struggles the most.

The table below summarizes the accuracy obtained, quantified using a L_2 norm, with the different α and number of support points

α	N_{sup}	L_2 error
1	0	$6.06 * 10^{-4}$
	10	$3.20 * 10{-}4$
	20	$3.18 * 10{-}4$
	40	$3.48 * 10{-}4$
2	0	$3.23 * 10^{-1}$
	10	$2.44 * 10^{-1}$
	20	$2.72 * 10^{-1}$
	40	$2.55 * 10^{-1}$

Table 5: L_2 error with α the frequency and N_{sup} the number of support points

6 Conclusion

In summary, PINNs perform well for most one and two dimensional problems but fail for predicting high frequency solutions and points outside the training domain. The model also struggles to predict solutions for three dimensional problems, forcing us to add supervised support points to help convergence.

Extending on our observation that support points help a model converge, it might be possible, for some applications, to generate support points naturally through empirical data (without knowledge of the exact solution). For example, when trying to model a heat diffusion process, a thermometer could provide temperature readings at a fixed point over time to help the model converge.

An improvement we'd suggest exploring for further study is the implementation of adaptive learning. The model can be trained recursively, by adjusting the distribution of collocation points according to regions of highest loss at the end of a run, then repeating the training process. This should lead to better resource allocation for training.

References

[1] ck-12. Numerical methods for solving odes, 2016. URL https://flexbooks.ck12.org/cbook/ck-12-calculus-concepts/section/8.14/primary/lesson/numerical-methods-for-solving-odes-calc/. Accessed: 18-12-2022.

[2] B. Dacorogna and C. Tanteri. *Analyse avancée pour ingénieurs*. Presses polytechniques et universitaires romandes, second edition, 2011.

[3] J. Kim, K. Lee, D. Lee, S. Y. Jin, and N. Park. Dpm: A novel training method for physics-informed neural networks in extrapolation, 2020. URL https://arxiv.org/abs/2012.02681. Accessed: 18-12-2022.

[4] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part 2): Data-driven discovery of nonlinear partial differential equations, 2017. URL https://arxiv.org/abs/1711.10566. Accessed: 18-12-2022.

[5] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part 1): Data-driven solutions of nonlinear partial differential equations, 2017. URL https://arxiv.org/abs/1711.10561. Accessed: 18-12-2022.

[6] M. Raissi, P. Perdikaris, and G. 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. ISSN 0021-9991. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Appendix

6.1 Effect of the collocation point sampling method for a one-dimensional Cauchy problem

From the papers we read on PINNs, we noticed that they always choose randomly spaced collocation points. The risk with uniform collocation points, when we know nothing about the solution format, is that we might pick points that don't represent the function domain well. For example if we want to train a problem that has a sinus solution and pick points: $0, \pi, 2\pi, ...$, then those points will always evaluate to zero. So using a uniform sampling method with points that match the resonance frequency of our solution will lead to a very imprecise model.

We wanted to put this theory to the test, so we solved the one-dimensional Cauchy problem with a simple $y(t) = sin(t)$ solution and tested whether the uniform or random collocation points would perform better. Unfortunately this restricted us to a single collocation point per half period which wasn't enough to train either model. Both results are bad and therefore weren't included in the main report.

The silver lining however, is that we can observe from figure 6, that in the second half of the domain the uniform points predict a straight line while the random points do make an effort to train to the data. Note that the uniform points try to train the sinus in the first part of the domain because the model is given the initial condition $y(0)$ and $y'(0)$ so has an intuition that the result is not a horizontal line, however as it moves away from the origin it will attempt to train to a horizontal line as that is the only information we give it.

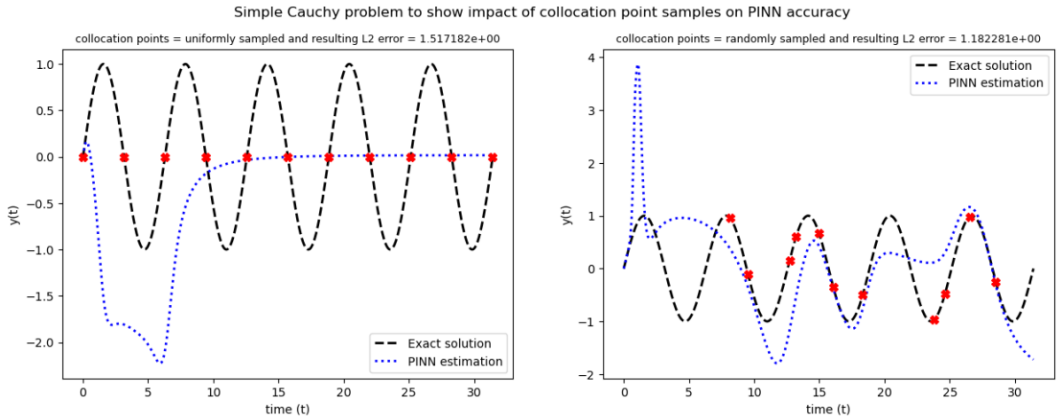


Figure 6: Predictions of the solution for a simple Cauchy problem using different collocation point sampling. The left graph uses uniform sampling while the right graph uses random sampling.

6.2 Viscous Burgers' equation

6.2.1 Physics Background

The viscous burgers' equation is a nonlinear partial differential equation that describes the evolution of a viscous fluid in a one space dimension. It is used to model various phenomena in fluid dynamics, such as shock waves, traffic flow, and fluid mixing. The viscous burgers' equation can be written as:

$$\frac{\partial u(t,x)}{\partial t} + u(t,x)\frac{\partial u(t,x)}{\partial x} = \nu \frac{\partial^2 u(t,x)}{\partial x^2}$$

$u(x,t)$ the velocity of the fluid, t is time, x is position, and ν is the coefficient of viscosity. The non linearity comes from $u(t,x)\frac{\partial u(t,x)}{\partial x}$. In the following sections we aimed to reproduce the burgers' equation inference and parameters identification performed in [5] and [4] using PyTorch instead of Tensorflow.

6.2.2 Inference

We will use the same setup as [5], which is:

$$\frac{\partial u(t,x)}{\partial t} + u(t,x)\frac{\partial u(t,x)}{\partial x} - \frac{0.01}{\pi}\frac{\partial^2 u(t,x)}{\partial x^2} = 0,\ x \in [-1,1].\ t \in [0,1]$$

With Dirichlet boundary conditions and an initial condition:

$$u(t,-1) = u(t,1) = 0,\ u(0,x) = -sin(\pi x)$$

Let us define $f(t,x) = u_t + uu_x - (0.01/\pi)u_{xx}$, the PDE approximated by the PINN, ideally we want $f(t,x)$ as close as possible to 0 for all (t,x) in the domain. The input of the PINN would be an input pair (t,x) and the output of the neural network would be $u_{approx}(t,x)$. In order to learn $u(t,x)$ that satisfies the nonlinear PDE, the boundaries and initial conditions, the PINN must minimize the following loss:

$$Loss = MSE_{bc_1} + MSE_{bc_2} + MSE_{ic} + MSE_f$$

The individual losses are:

$$\begin{cases} MSE_{bc_1} = \frac{1}{N_{bc_1}} \sum_{i=1}^{N_{bc_1}} |u(t^i_{bc_1}, -1) - 0|^2 \\ MSE_{bc_2} = \frac{1}{N_{bc_2}} \sum_{i=1}^{N_{bc_2}} |u(t^i_{bc_2}, 1) - 0|^2 \\ MSE_{ic} = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |u(0, x^i_{ic}) - (-sin(\pi x^i_{ic}))|^2 \\ MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t^i_f, x^i_f)|^2 \end{cases}$$

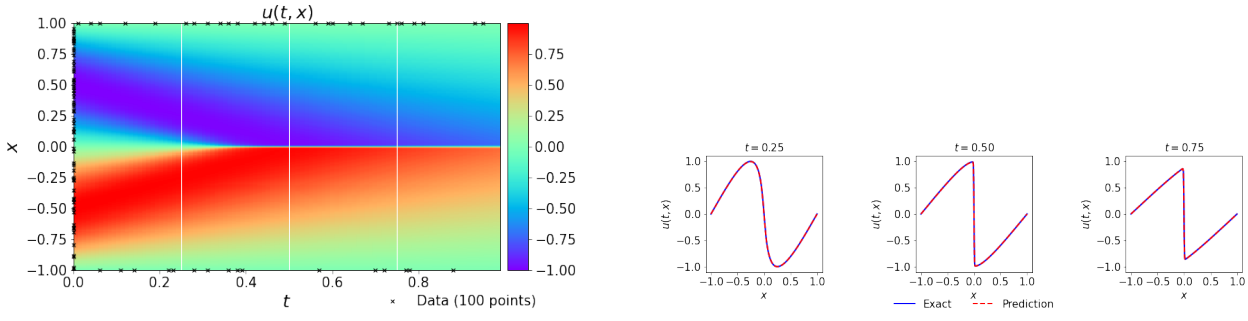
The sets $\{t^i_{bc_1}, -1\}_{i=1}^{N_{bc_1}}$, $\{t^i_{bc_2}, 1\}_{i=1}^{N_{bc_2}}$, $\{0, x^i_{ic}\}_{i=1}^{N_{ic}}$ form the boundary and initial training data on $u(t,x)$. The set $\{t^i_f, x^i_f\}_{i=1}^{N_f}$ represents the physics informed part of the PINNs, these points are collocations points, they are used to enforce the structure of the PDE, without them it would be a classical neural network and the output would not correctly satisfy the PDE. MSE_f can be seen as L_2 regularization, it will penalize solutions that do not satisfy the PDE.

The boundary and initial training data were obtained from a file called *burgers_shock.mat*, which contains a numerical approximation of the exact solution for the Burgers' equation on a discrete grid. The time space is discretized into 100 equally spaced points with a step size of $\Delta t = 0.01$, and the space space is discretized into 256 equally spaced points with a step size of $\Delta x = 0.00784$. We used the same t and x, but we decided to use the exact values of $u(x,t)$ for the boundary and initial conditions unlike in [5], where the numerical approximations were used. By using this grid, we get 100 $\{t_{bc}, x_{bc}, u_{bc}\}$ pairs per boundary condition and 256 pairs for the initial condition, making in total 456 pairs. Then, we generate $N_f = 10000$ collocations points using a *Latin Hypercube Sampling* strategy, which aims to evenly distribute the points within the domain, while also minimizing the correlation between them. To these 10000 collocation pairs we add the previous 456 $\{t_{bc}, x_{bc}\}$ pairs to improve accuracy, and from the 456 $\{t_{bc}, x_{bc}, u_{bc}\}$ pairs, we randomly select $N_u = 100$ to be our boundary and initial conditions training data.

The network is composed of 9 fully connected hidden layers, all having 20 nodes and a hyperbolic tangent activation function. The input is of size 2, representing a pair $\{t,x\}$, and the output of size representing $u(t,x)$, where u is the approximation of the true solution by the PINN. The weights were initialized using Xavier initialization to prevent vanishing or exploding gradients, and improve the convergence rate. We choose LBFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) over Adam as optimizer, because it requires fewer function evaluations and gradient evaluations per iteration, making it converges faster and gave more accurate results. Also LBFGS could be prefered for large-scale optimization problems such as solving nonlinear PDE with a high number of collocations points. At each iteration, all the training data does a forward pass in the network, and then the association u is used to compute the loss.

After 5600 iterations, LBFGS converges, and we got $MSE_u = MSE_{bc_1} + MSE_{bc_2} + MSE_{ic} = 1.35 \times 10^{-7}$ and $MSE_f = 5.30 \times 10^{-6}$. After, we compared the model predictions to the numerical approximations in the *burgers_shock.mat* file, and got $MSE_u = 5.23 \times 10^{-7}$ and $MSE_f = 5.97 \times 10^{-6}$.The following figure 7 shows the predicted solution by the PINN, and that the PINN correctly learned the non-linearity of the viscous burgers' equation.

Finally we performed two systematic studies. The first one, we increasingly varied the number of neurons and layers with $N_u = 100$ and $N_f = 10000$ and compared the models predictions to the numerical approximations of *burgers_shock.mat*. As expected, the smallest errors were obtained on the model with highest explicative power, the highest number of layers and neurons. The results can be found in the following table 1. The second study, we increasingly varied the number of initial and boundary pairs N_u and the collocations pairs N_f, with a PINN of 9 hidden layers and 20 neurons. Similarly, the model performed better with more data, the results can be found in the following table 2.



(a) Estimated solution $u(t,x)$

(b) Evolution of the non-linearity

Figure 7: Viscous burgers' equation inference

6.2.3 Identification

The viscous burgers' equation can also be written as:

$$\frac{\partial u(t,x)}{\partial t} + \lambda_1 u(t,x)\frac{\partial u(t,x)}{\partial x} - \lambda_2 \frac{\partial^2 u(t,x)}{\partial x^2} = 0$$

where λ_1 and λ_2 are unknown real parameters to estimate using PINNs. In parameters identification, it is assumed that the $u(t,x)$ is known, and we want to find the PDE's parameters. From the previous setup, we know that the true parameters are $\lambda_{1_{true}} = 1$ and $\lambda_{2_{true}} = \frac{0.01}{\pi}$.

Let us define $f(t,x) = u_t + \lambda_1 u u_x - \lambda_2 u_{xx}$, the unknown PDE the PINN will approximate, we want it to be as close as possible to 0 for all (t,x) in the domain. The loss can defined as:

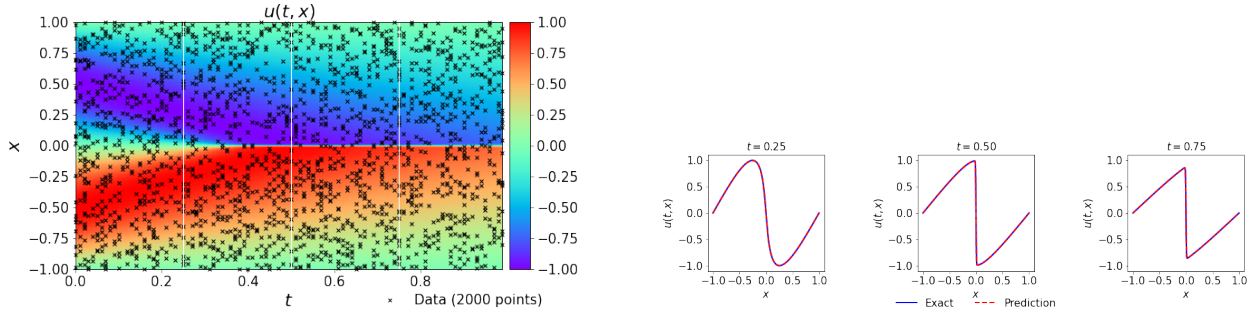
$$Loss = MSE_u + MSE_f$$

In this problem, we do not consider the boundary and initial conditions as $u(t,x)$ is supposed to be known. The individual losses are:

$$\begin{cases} MSE_u = \frac{1}{N}\sum_{i=1}^{N} |u(t_u^i, x_u^i) - u^i|^2 \\ MSE_f = \frac{1}{N}\sum_{i=1}^{N} |f(t_u^i, x_u^i)|^2 \end{cases}$$

Where $\{t_u^i, x_u^i, u^i\}_{i=1}^N$ is the training data set on $u(t,x)$. The pairs are obtained with from *burgers_shock.mat*, in total the file contains 25600 pairs, we randomly select $N = 2000$ pairs. We used the same PINN architecture as in inference, and we added to the model parameters λ_1 and λ_2. When calculating MSE_f at each iteration, the model will update λ_1 and λ_2 to minimize the loss, as $\lambda_{2_{true}} =\approx 0.00318$ is very small, during the calculation of f we will use e^{λ_2} instead of λ_2 to make the scale more appropriate. After 2900 iterations, the LBFGS optimizer converged, and we got $MSE_u = 1.10\times10^{-5}$ and $MSE_f = 2.33\times10^{-5}$, and comparing to the numerical approximations of the file we got $MSE_u = 1.23\times10^{-5}$ and $MSE_f = 1.17\times10^{-3}$. The prediction $u(t,x)$ can be seen in the following figure 8. The estimated parameters are $\lambda_1 = 0.9959$ and $\lambda_2 = 0.0032$, which gives us a relative error of 0.4066% for λ_1 and 0.8525% for λ_2.

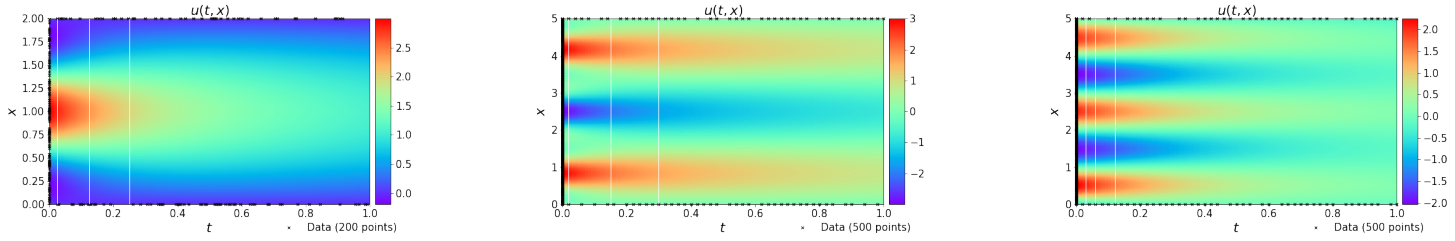
As in [4], we perform two systemic studies. The first one, we keep $N = 2000$ and we vary the number of layers and neurons and look for the smallest relative errors on λ_1 and λ_2. From table 3, we observe that models with high explication power have smaller relative errors on the parameters. The second one, we add different levels of uncorrelated noise to u^i used in MSE_u and observe the respective relative errors on the parameters. We can see that up to 10% the relative error with respect to λ_1 remains small 1%, but for high noise level, the error becomes very high on λ_2. This is because λ_2 multiplies the second derivative term in the PDE, making it very sensible to variations and noise, as we can see in table 4.



(a) Estimated solution $u(t,x)$ (b) Evolution of the non-linearity

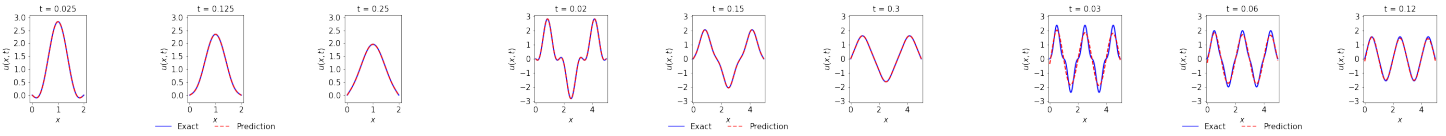
Figure 8: Viscous burgers' equation identification

6.3 Heat equation 2D



(a) $f = 1$ (b) $f = 3$ (c) $f = 5$

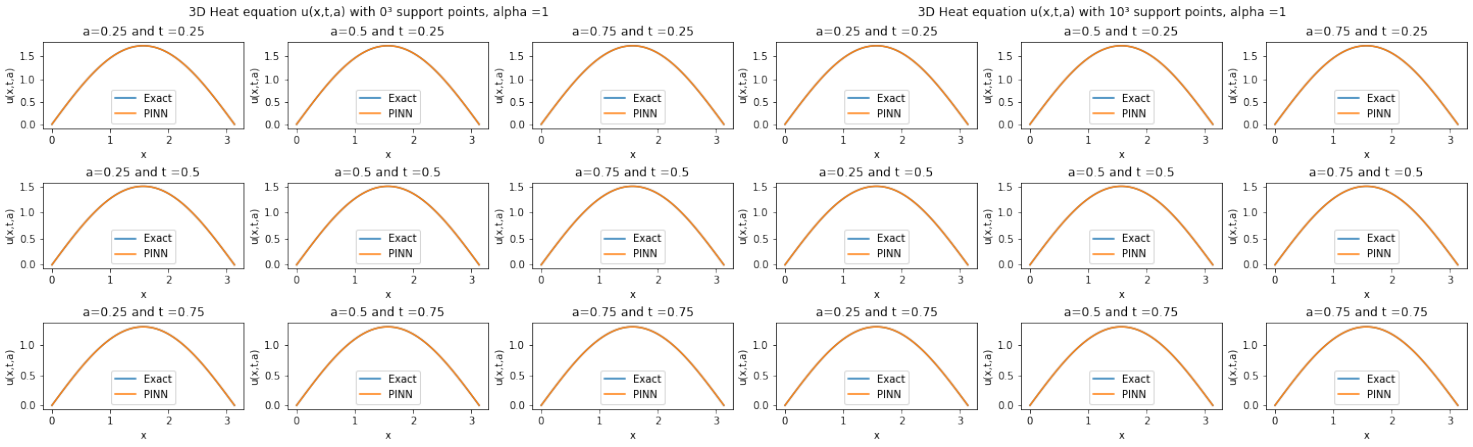
Figure 9: Heat equation 2D plot of $u(t,x)$ for different frequencies



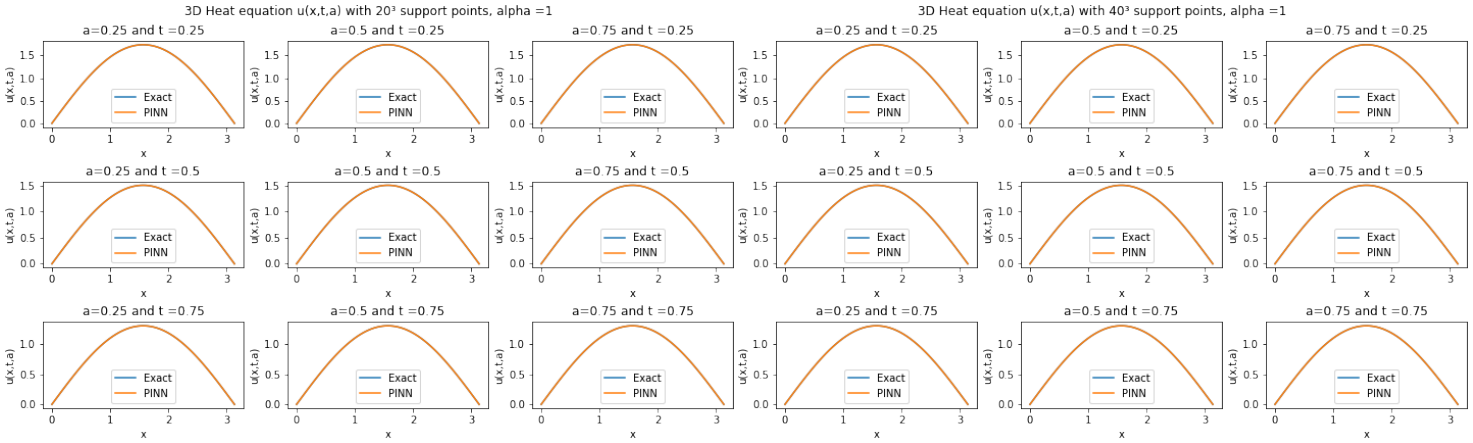
(a) $f = 1$ (b) $f = 3$ (c) $f = 5$

Figure 10: Heat equation 2D time evolution for different frequencies

6.4 Heat equation 3D



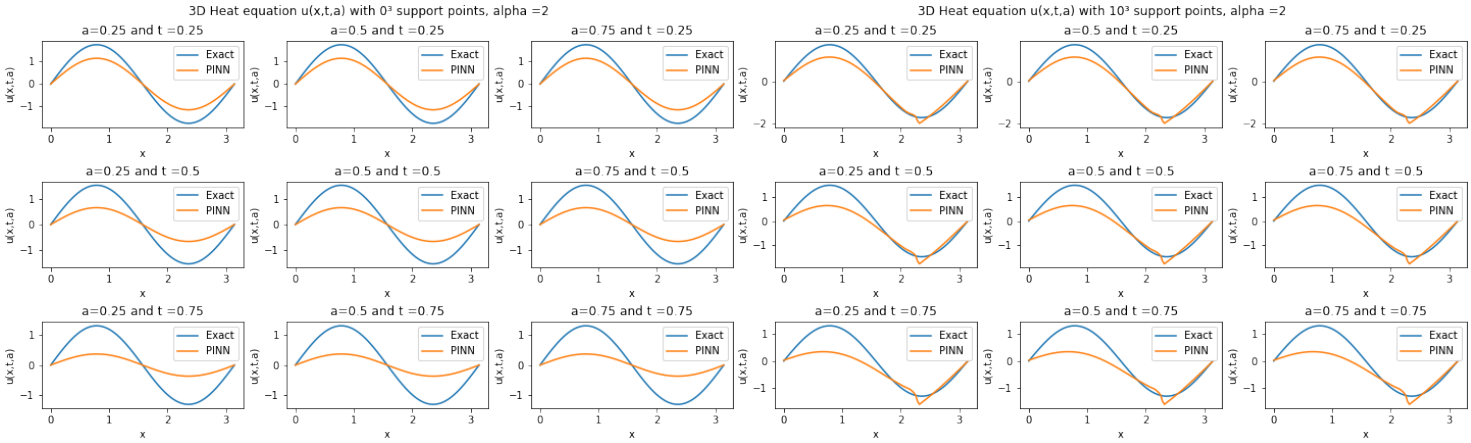
(a) Estimated solution $u(x, t, a)$ with 0 support points, α =11 (b) Estimated solution $u(x, t, a)$ with 10 support points, α =1



(c) Estimated solution $u(x, t, a)$ with 20 support points, α =1 (d) Estimated solution $u(x, t, a)$ with 40 support points, α =1

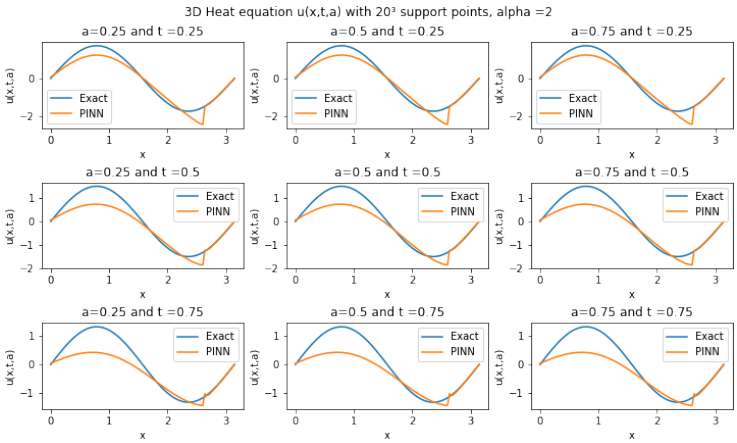
Figure 11: Predicted solutions for α =1

Figure 11 displays the predicted solution with [0, 10, 20, 40] support points for α=1. The match is almost perfect.

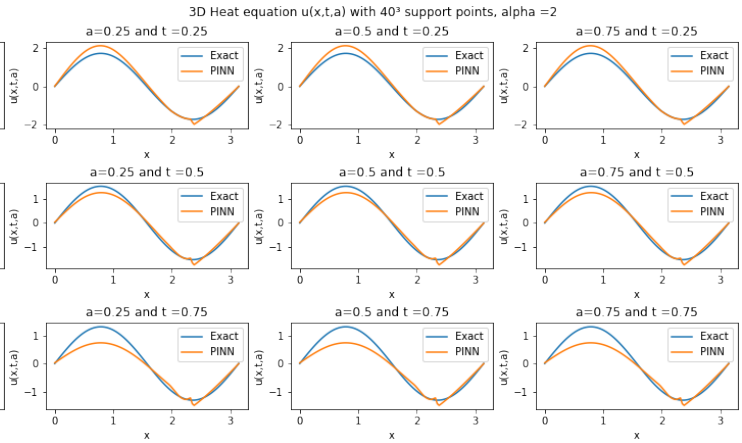


(a) Estimated solution $u(x, t, a)$ with 0 support points, α =2

(b) Estimated solution $u(x, t, a)$ with 10 support points, α =2



(c) Estimated solution $u(x, t, a)$ with 20 support points, α =2



(d) Estimated solution $u(x, t, a)$ with 40 support points, α =2

Figure 12: Predicted solutions for α =2

Figure 12 displays the predicted solution with [0, 10, 20, 40] support points for α=2. The PINN struggles more to find the true solution, and artifacts can be observed.