Exploring the applicability of neural networks to solve ODEs and PDEs

Adam Jakobsen and Nils Taugbøl[*]
University of Oslo, Department of Physics
(Dated: January 13, 2023)

We explore the flexibility of neural networks for solving ordinary and partial differential equations. The neural networks are compared to the forward time centered space method for solving the one-dimensional diffusion equation and to a standard eigenvalue solver when solving the eigenvalue problem as an ODE. We find that neural networks are highly applicable for solving PDEs such as the diffusion equation as they are not bound by stability criteria and produce a function as a solution. Similarly we find neural networks to be applicable for solving the eigenvalue problem, though we find some discrepancies between our results and the theory presented by Yi et. al [1].

Keywords: Eigenvalue, Neural Network, Partial Differential Equations

I. INTRODUCTION

Partial differential equations (PDEs) are a fundamental tool in many areas of science and engineering. They often provide critical information to understand and predict the behavior of various systems. Some examples are the Schrödinger equation, which describes the behaviour of quantum mechanical systems and the Navier-Stokes equations, which are used to model the flow of fluids.

Traditional methods for solving PDEs, such as finite difference methods are highly accurate and easy to implement but are limited by the fact that they involve replacing a continuous domain with a discretized grid. Consequently, the results are also descretized.

The universal approximation theorem, first proven by George Cybenko in 1989 [2], states that a feedforward neural network with a single hidden layer and a finite number of nodes can approximate any continuous function to any desired degree of accuracy. In recent years, there has been a growing interest in using neural networks as an alternative approach for solving PDEs. One of the main advantages is that neural networks offer a solution in the form of a function that can be evaluated continuously.

Another practical, yet similar application of neural networks is for solving eigenvalue problems. These problems occur in various fields, and involve finding the eigenvalues and corresponding eigenvectors of a matrix or operator which satisfy specific equations. In 2003, Yi et al. presented a method for solving eigenvalue problems using neural networks [1]. The approach sets the eigenvalue problem up as an ordinary differential equation and so can be solved as such by a neural network.

In this report, we explore and compare the use of the forward Euler explicit scheme, known also as the forward time centered space (FTCS) method, and a neural network to solve the one-dimensional diffusion equation. We investigate the solution at two time points using different discretizations maintaining the stability criterion of the explicit scheme and compare to the solution using a neural network. We then follow the work of Yi et al. to find the eigenvalues and eigenvectors of a real symmetric matrix using a neural network compared to a standard eigenvalue solver.

The report is structured as follows. First, in Section II, we introduce the necessary background on the diffusion equation and eigenvalue problem. We also derive the FTCS method and present the neural network architecture which we will apply. Then, in Section III, we present our results of the methods used to solve the diffusion equation as well as those for the eigenvalue problem. Thereafter, in Section IV, we discuss the results and compare the methods. Finally, in Section V, we conclude by summarizing the advantages and disadvantages of solving PDEs and eigenvalue problems using neural networks.

II. METHOD

A. The diffusion equation

In this part we will present the necessary background knowledge for solving the diffusion equation, using two different numerical approaches. We will use the forward time centered space method and a neural network.

1. *The diffusion equation*

The diffusion equation in a one-dimensional homogeneous medium is given by the partial differential equation

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

[*] Repository:https://github.com/Adamjakobsen/FYS-STK4155

where $f(x,t)$ is the diffusive quantity, D is the diffusion coefficient, t is the time coordinate and x is the spacial coordinate [3]. In this report we use $D = -1$, which corresponds to the heat equation. From (1) we wish to find $f(x,t)$ using numerical methods. We will study a dimensionless system where we set the spacial domain

$$x \in [0,1],$$

where our initial condition is

$$f(x,0) = \sin \pi x$$

and where we apply Dirichlet boundary conditions

$$f(0,t) = f(1,t) = 0,$$

for $t \geq 0$.

The exact solution of (1) is given by

$$f_{\text{exact}}(x,t) = \sin(\pi x) e^{-\pi^2 t},$$

which we will use to evaluate our numerical solutions.

2. *Forward time centered space*

A natural numerical approach to solving the diffusion equation is by use of the forward time centered space method. This method is a simple finite difference method to solve ordinary or partial differential equations, which may be non-linear. The general procedure is to discretize the time and spacial dimensions which allows us to approximate both sides of (1) and solve for one time step at a time.

We discretize these coordinates by introducing a step size in the spacial dimension Δx and in thetime dimension Δt as follows

$$\begin{aligned} x_i &= i\Delta x,\ i \in \{0,1,2,...,i_{\max}\} \\ t_n &= n\Delta t,\ n \in \{0,1,2,...,n_{\max}\}, \end{aligned}$$

where $i_{\max} = 1/\Delta x$ and $n_{\max} = T/\Delta t$ for a total time T. Using the indices i and n we can describe $f(x,t)$ at each point x_i and t_n as f_i^n. The left hand side of (1) can be approximated using the forward difference scheme, also known as forward Euler,

$$\frac{\partial f_i^n}{\partial t} = \frac{f_i^{n+1} - f_i^n}{\Delta t} + \mathcal{O}(\Delta t). \tag{2}$$

By truncating before the second term we then get an approximation with an error scaling with $\mathcal{O}(\Delta t)$. The right hand side can be approximated using the second order central difference

$$\frac{\partial^2 f_i^n}{\partial x^2} = \frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{(\Delta x)^2} + \mathcal{O}\left((\Delta x)^2\right), \tag{3}$$

where we again obtain an approximation by truncating away the second term. Note that the error of the approximation of the RHS scales with $\mathcal{O}((\Delta x)^2)$. Inserting equations (2) and (3) into (1) and solving for f_i^n we then get

$$f_i^{n+1} = f_i^n + \frac{\Delta t}{(\Delta x)^2}\left(f_{i+1}^n - 2f_i^n + f_{i-1}^n\right),$$

which can easily be solved numerically. Note that to evolve from one time step to another only the current time step is necessary, this makes the the scheme explicit.

The forward time centered space algorithm must fulfill the stability criteria [4]

$$\left| D\frac{\Delta t}{(\Delta x)^2} \right| \leq \frac{1}{2}.$$

Note that the stability criteria ensures that our method will not diverge from the exact solution, however the method will still accumulate a global error. The algorithm for the forward time centered space method can be found in Algorithm 1.

Algorithm 1: Forward Time Centered Space

```
$\mathbf{f}^0 = (x_0, x_1, ..., x_{i_{\max}}$        ▷ Set up the initial state
$\Delta x$                                  ▷ Choose spacial step size
$\Delta t = (\Delta x)^2/2$  ▷ Set highest possible time step **for**
$T/\Delta t$ *and all* $i$ **do**
    **end**
    $f_i^{n+1} = f_i^n + \frac{\Delta t}{(\Delta x)^2}(f_{i+1}^n - 2f_i^n + f_{i-1}^n)$        ▷
Evolve the model
```

Following the truncation done in (2) and (3) we have a global error that scales with the magnitude of the step sizes as

$$\varepsilon \propto \mathcal{O}\left((\Delta x)^2\right)\mathcal{O}(\Delta t). \tag{4}$$

3. *Neural network*

Another, drastically different, approach to solving the diffusion equation numerically is by the use of neural networks. Neural networks are described in detail by Jakobsen et. al [5]. The neural network has to optimise its weights and biases, contained in a matrix P, based on an initial trial solution by minimising a cost function. The trial solution has to satisfy the conditions set upon the system. A generic trial function can be expressed as

$$f_{\text{trial}}(x,t) = h_1(x,t) + h_2(x,t)N(x,t,P),$$

where $h_1(x,t)$ and $h_2(x,t)$ ensures that $f_{\text{trial}}(x,t)$ satisfies the conditions and $N(x,t,P)$ is the neural network.

Table I: Architecture chosen for solving the one-dimensional diffusion equation using a neural network.

Layer	Activation function
Hidden 1	tanh
Hidden 2	sigmoid
Output	None

The initial condition can be enforced through

$$h_1(x,t) = (1-t)\sin(\pi x),$$

and to fulfill the Dirichlet boundary condition we set

$$h_2(x,t) = x(1-x)t.$$

Our trial function then becomes

$$f_{\text{trial}}(x,t) = (1-t)\sin(\pi x) + x(1-x)tN(x,t,P).$$

The cost function could, in this case, compare the output of the neural network to the exact result when optimising its parameters. However, this would defeat the purpose of using a neural network to solve a partial differential equation. Therefore we will use the mean of the squared residual as a cost function which is expressed as

$$\langle r^2 \rangle = \left\langle \left(\frac{\partial f_i^n}{\partial t} - \frac{\partial^2 f_i^n}{\partial x^2} \right)^2 \right\rangle,$$

where we compare both sides of (1) and want is to have $r = 0$. This cost function translates to a minimisation problem which the neural network can use to learn without the need for computing expected outputs or using exact solutions. This cost function requires the calculation of a first and second derivative, we compute these using automatic differentiation [6].

The architecture chosen for this particular problem can be found in Table I with 100 nodes for the hidden layers. We have chosen a learning rate $\eta = 0.01$ with 5000 epochs and use the ADAM optimisation method. These hyperparameters were found to yield satisfying results through trial and error.

B. Eigenvalues

Let $A \in \mathbb{R}^{n\times n}$ be a real symmetric matrix. We then have that an eigenvector $v \in \mathbb{R}^n$ with eigenvalue, λ, must satisfy

$$Av = \lambda v.$$

An approach to computing the eigenvalues and eigenvectors of a symmetric matrix A has been proposed by Yi et al.[1]. The basis of the method is to define the eigenvectors as solutions to an ordinary differential equation given by

$$\frac{dx(t)}{dt} = -x(t) + f(x(t)), \qquad (5)$$

where $f(x(t))$ is given by

$$f(x(t)) = \left(x^T x A + (1 - x^T A x) I\right) x,$$

where $x \in \mathbb{R}^n$, and $I \in \mathbb{R}^{n\times n}$ is the identity matrix.

The vector x is said to be an equilibrium point of (5) if $-x + f(x) = 0$. Thus, it follows that

$$x^T x A x - x^T A x x = 0,$$

if x is an eigenvector , i.e $\lim_{t\to\infty} x(t) = v$, or if x is the null vector. We then get

$$\lambda = \frac{v^T A v}{v^T v} v. \qquad (6)$$

We state the following Theorem, which is a combination of Theorems 3-4 in [1], where the proofs are provided.

Theorem 1. *Every solution of (5) starting from any $x(0) \neq \boldsymbol{0}$ will converge to an eigenvector of A.*

- *Given a nonzero $x(0) \in \mathbb{R}^n$ that is not orthogonal to the eigenspace of the largest eigenvalue, then $x(t)$ converges to an eigenvector of A corresponding to the largest eigenvalue.*

- *Given a nonzero $x(0) \in \mathbb{R}^n$ that is not orthogonal to the eigenspace of the smallest eigenvalue, replacing $A \to -A$ results in $x(t)$ converging to an eigenvector of A corresponding to the smallest eigenvalue.*

1. Neural Network eigenvalue problem

In order to apply the method for finding the eigenvalues and eigenvectors of a symmetric matrix to a neural network, we need to define a trial function, $g(t,P)$, which satisfies the following conditions:

- $g(0,P) = x(0)$
- $\lim_{t\to\infty} g(t,P) = \lim_{t\to\infty} N(t,P)$

Here the first condition ensures that the trial function is a solution of the ODE at $t = 0$ and the second condition ensures that the trial function converges to the solution of the ODE. Our chosen trial function is then

$$g(t,p) = e^{-t}x(0) + (1 - e^{-t})N(t,P).$$

Table II: Architecture chosen for solving the eigenvalue problem using a neural network.

Layer	Nodes	Activation function
Hidden 1	1000	ReLU
Output	6	None

The cost function can then be defined as

$$C(t, P) = \left\langle \left(\frac{dg(t_i, P)}{dt} + g(t_i, P) - f\left(g(t_i, P)\right) \right)^2 \right\rangle ,$$

where $f(g(t, p))$ is defined by (6).

The architecture chosen for this particular problem can be found in Table II. We have chosen a learning rate $\eta = 0.001$ using the ADAM optimisation method. These values were found to yield satisfying results through trial and error.

III. RESULTS

A. The diffusion equation

As the setup of the diffusion equation in the previous section set up the problem using dimensionless variables, the partial differential equations were solved for a total time $T = 1$ whilst varying Δx and Δt. The diffusion equation starts as a half-period sin-wave as dictated by the initial condition and flattens toward the equilibrium.

For the FTCS method we computed the solution using two different spacial step sizes $\Delta x = \{0.1, 0.01\}$ where Δt followed the stability criteria. The solution with $\Delta x = 0.1$ at a few selected timestamps can be seen in Figure 1. The same solution when using $\Delta x = 0.01$, and thereat $\Delta t = 0.00005$, can be seen in Figure 2. The MSE as a function of time from the solutions of the FTCS method using the two different Δx are seen in Figure 7. The MSE was calculated using the exact solution.

The first approach using the neural network was to scan for an appropriate learning rate and number of epochs, we did not include theis search as these methods are well documented by Jakobsen et. al [5]. We found that a learning rate $\eta = 0.01$ and 5000 epochs using the ADAM optimisation method produced acceptable results, the architecture can be seen in Table I. We applied step sizes dictated by the stability criteria for the FTCS method and found those results satisfactory, these results are shown in Figure 3. To explore the flexibility of the NN we also solved the diffusion equation using equal step sizes in both time and space $\Delta x = \Delta t = 0.02$, these results are shown in Figure 4.

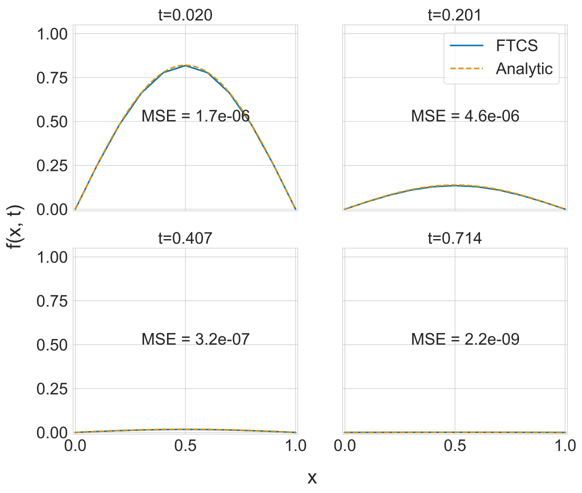


Figure 1: Comparison of the exact solution of the diffusion equation to the solution from the FTCS method. Here the chosen step sizes were $\Delta x = 0.1$ and $\Delta t = 0.005$. The chosen timestamps are $t = \{0.02, 0.2, 0.4, 0.7\}$. The MSE at each timestamp can be seen in their respective frame.

As the NN learns through a forward- and backpropagation process we included the learning history when solving the diffusion equation with different step sizes in Figure 5. Figure 6 shows the scaled solutions corresponding to the bottom right of Figures 2 and 4. The scaled solutions allow us to observe the difference between the the methods' approaches to solving the PDE.

The MSE of both methods' solutions to the diffusion equation can be seen in Figure 7 where we have included the MSE of the neural network trained with a varying number of epochs. Once the NN was trained we tested it with different values for Δx and set $\Delta x = \Delta t$. The MSE of these tests and of the solution with the original step sizes are shown in Figure 8

To better compare the neural network and the FTCS method we have included the average computation time required for each method performed three individual times in Table III. We included the training time for the neural network and the computation time required for a model trained with 5000 epochs with step sizes $\Delta x = 0.1$ and $\Delta t = 5 \times 10^{-3}$ to produce solutions with different step sizes.

B. Eigenvalue problem

In order to compute the eigenvalues and eigenvector, a symmetric 6×6 matrix, A, was generated from

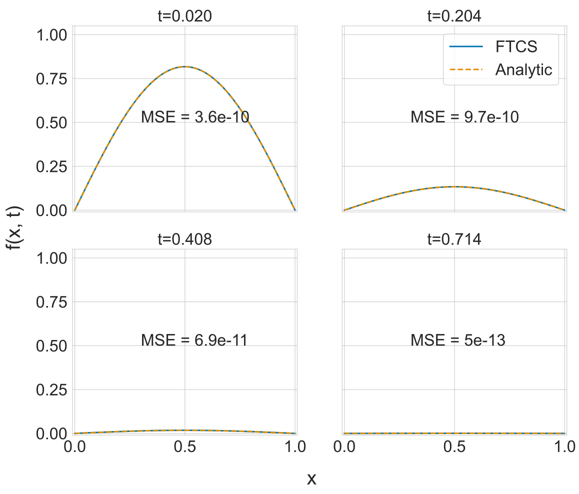


Figure 2: Comparison of the exact solution of the diffusion equation to the solution from the FTCS method. Here the chosen step sizes were $\Delta x = 0.01$ and $\Delta t = 5 \times 10^{-5}$. The chosen timestamps are $t = \{0.02, 0.2, 0.4, 0.7\}$. The MSE at each timestamp can be seen in their respective frame.

Table III: Average computation time required for the different methods. The computation time was calculated over an average of three independent computations.

Method	Δx	Δt	Time (s)
FTCS	0.1	0.005	0.003
FTCS	0.01	5×10^{-5}	2.609
NN 3000 epochs	0.1	5×10^{-3}	741.730
NN 5000 epochs	0.1	5×10^{-3}	1117.993
NN 7000 epochs	0.1	5×10^{-3}	1546.551
NN trained	3.3×10^{-2}	3.3×10^{-2}	2.135
NN trained	2.0×10^{-2}	2.0×10^{-2}	2.334
NN trained	1.4×10^{-2}	1.4×10^{-2}	2.171

a random uniform distribution, while the initial vector was drawn from a normal distribution $\mathcal{N}(0,1)$. The eigenvalues and eigenvectors were computed using a neural network with a single hidden layer with 1000 nodes. We found that a learning rate $\eta = 0.001$ using the ADAM optimization method yielded the best results when comparing to the solutions from standard numerical solver.The predictions were made with total time $T = 10^3$ with $N = 100$ points. The largest eigenvalue,λ_{max}, was computed using A, while the smallest eigenvalue,λ_{min}, was computed by letting $A \rightarrow -A$.These results are summarized in Table IV.

In Figure 9 we see the eigenvalue converging to λ_{max}

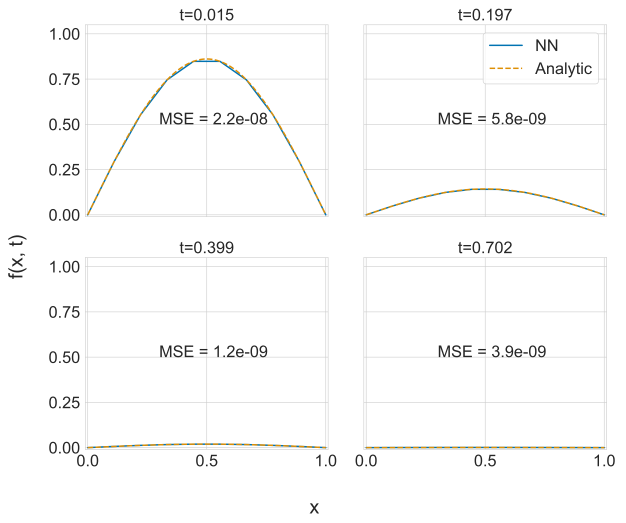


Figure 3: Comparison of the exact solution of the diffusion equation to the solution from the neural network. Here the chosen step sizes were $\Delta x = 0.1$ and $\Delta t = 5 \times 10^{-3}$. The chosen timestamps are $t = \{0.02, 0.2, 0.4, 0.7\}$. The MSE at each timestamp can be seen in their respective frame.

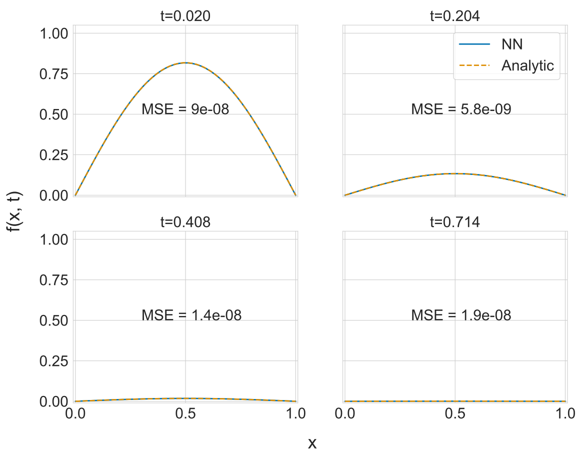
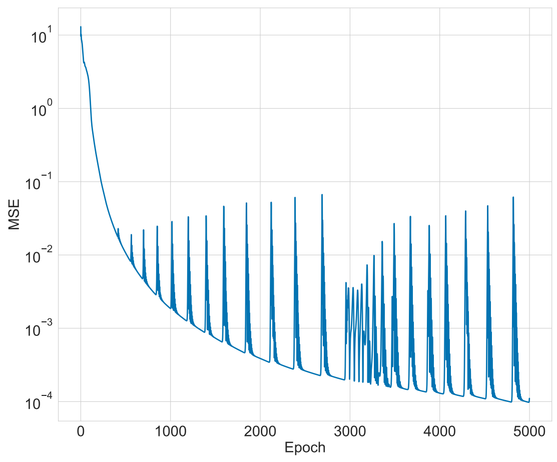


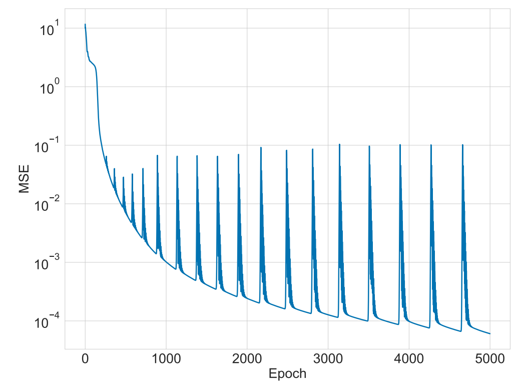
Figure 4: Comparison of the exact solution of the diffusion equation to the solution from the neural network. Here the chosen step sizes were $\Delta x = \Delta t = 0.02$. The chosen timestamps are $t = \{0.02, 0.2, 0.4, 0.7\}$. The MSE at each timestamp can be seen in their respective frame.

at approximately 1500 epochs, while the eigenvector components in Figure 11 converge after approximately 3000 epochs.

In Figure 10 we observe that the eigenvalue converges to λ_{min} slightly faster than for the largest eigenvalue,

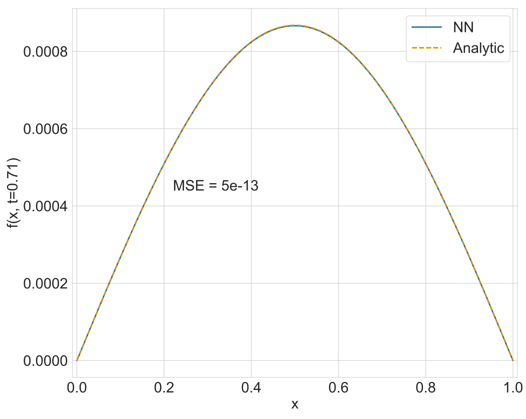


(a) Learning history of the neural network with step sizes $\Delta x = 0.1$ and $\Delta t = 5 \times 10^{-3}$.

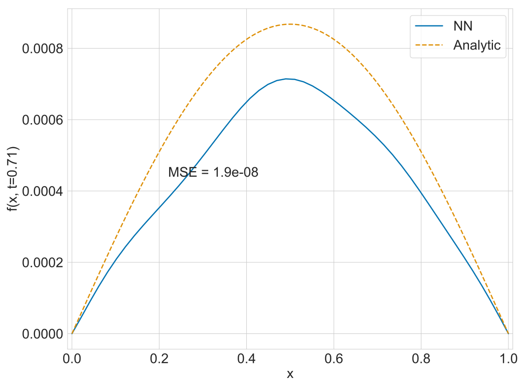


(b) Learning history of the neural network with step sizes $\Delta x = \Delta t = 0.02$.

Figure 5: Learning history of the neural network as a function of epochs. Figure 5a has step sizes following the stability ctriteria of the FTCS method. Figure 5b has equal step sizes in the spacial and time dimensions.



(a) Solution of the FTCS method with step sizes $\Delta x = 0.01$ and $\Delta t = 5 \times 10^{-5}$.



(b) Solution of the neural network with step sizes $\Delta x = \Delta t = 0.02$.

Figure 6: Scaled solutions from the FTCS method and neural network at $t = 0.7$. We see how the solutions' shape differ from the exact solution around the equilibrium point of the model. The MSE of the solutions can be seen in the figure frame.

while some of it's corresponding eigenvector components converge significantly faster.

Using the same symmetric matrix A, we compute the eigenvalues starting from the eigenvectors corresponding to λ_{max} and λ_{min}. In Figure 13 we see that our estimate is not able to converge to the largest eigenvalue when starting from its corresponding eigenvector. The same can be seen in Figure 14, which shows that when starting from the eigenvector corresponding to λ_{min} and letting $A \to -A$ our estimate converges to another eigenvalue.

IV. DISCUSSION

A. The diffusion equation

We can see from Figures 2, 1, 3 and 4 that both the FTCS and NN methods for their varying resolutions provide good fits to the exact solution. The methods solve the problem differently which we see in Figures 6a and 6b. The FTCS method predicts how the PDE evolves from one timestep to another and so the shape is very much like the exact solution. The NN, has a training process where it learns to fit the data as best

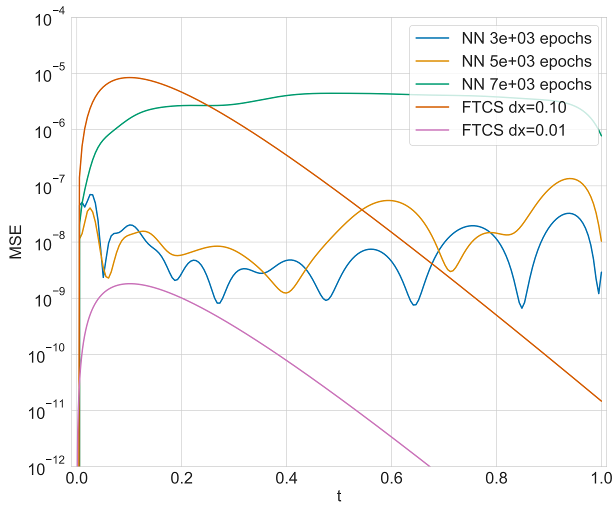


Figure 7: MSE of the solution to the diffusion equation as a function of time from the FTCS method and the neural network for different numbers of epochs. For the FTCS method we applied step sizes $\Delta x = 0.01$ and $\Delta t = 5 \times 10^{-5}$ and for the neural network we applie step sizes $\Delta x = 0.1$ and $\Delta t = 5 \times 10^{-3}$.

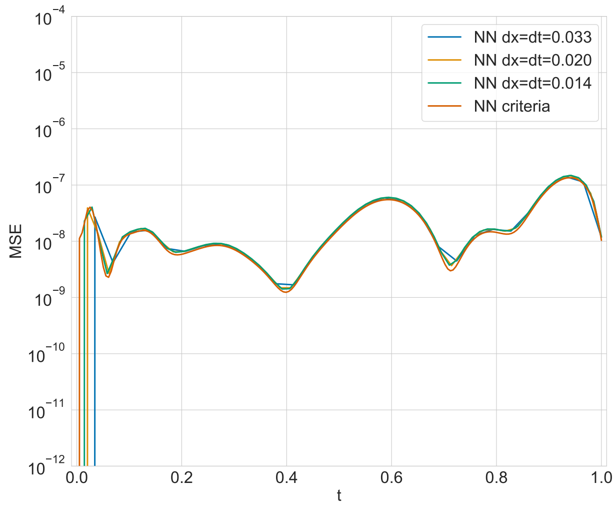


Figure 8: MSE of the solutions from the trained neural network. The network was trained using $\Delta x = 0.1$ and $\Delta t = 5 \times 10^{-3}$ and tested using different values of $\Delta x = \Delta t$.

Table IV: Estimate of the largest and smallest eigenvalue with relative errors when compared to `numpy` solution.

	`numpy`	NN	Relative Error
$\lambda_{\max}$	2.687	2.687	$1.2 \cdot 10^{-4}\%$
$\lambda_{\min}$	-0.700	-0.700	$3.9 \cdot 10^{-3}\%$

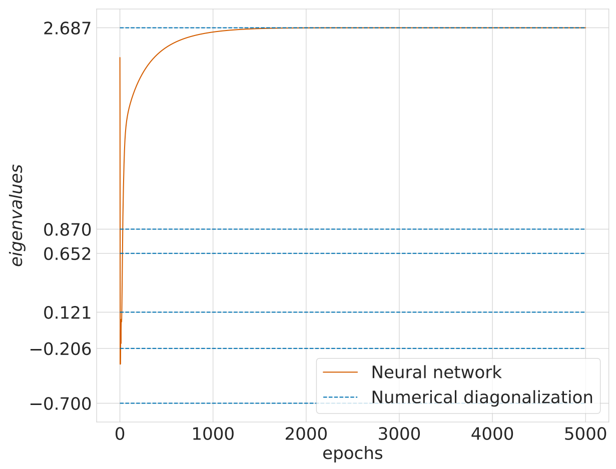


Figure 9: Estimate of the largest eigenvalue as a function of epochs using a neural network with a single hidden layer with 1000 nodes. The solution using `numpy`'s eigenvalue solver is shown as dashed lines.

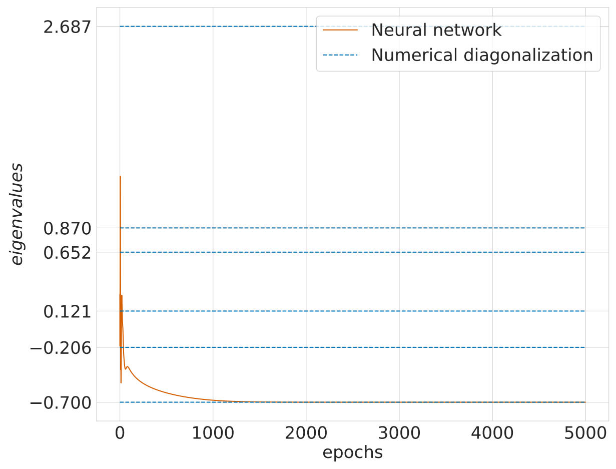


Figure 10: Estimate of the smallest eigenvalue, when $A \to -A$ as a function of epochs using a neural network with a single hidden layer with 1000 nodes. The solution using `numpy`'s eigenvalue solver is shown as dashed lines.

as possible and so the shape of the solution, though a good fit, varies somewhat from the exact solution.

We can see from Figure 7 that the FTCS method increases its accuracy when decreasing the step size. The increase in accuracy is by a magnitude 10^5 which is expected from (4). The initial condition is identical to the exact solution for $t = 0$ which explains why the MSE starts at zero for all methods and simulations. The diffusion equation has greater changes at earlier timestamps, in $t \in (0, 0.2]$, which corresponds to the

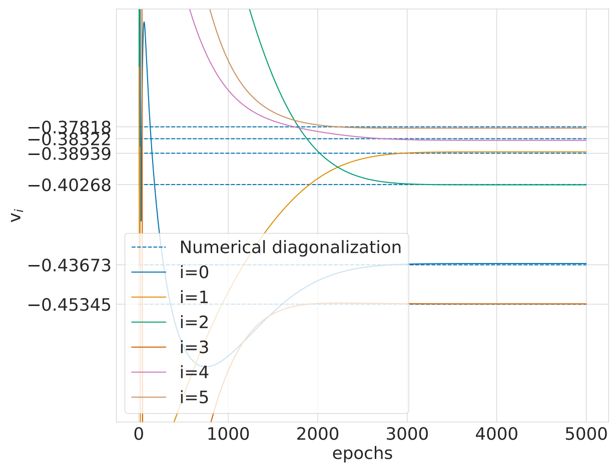


Figure 11: Estimate of the eigenvector components corresponding to the largest eigenvalue as a function of epochs using a neural network with a single hidden layer with 1000 nodes. The solution using `numpy`'s eigenvalue solver is shown as dashed lines.

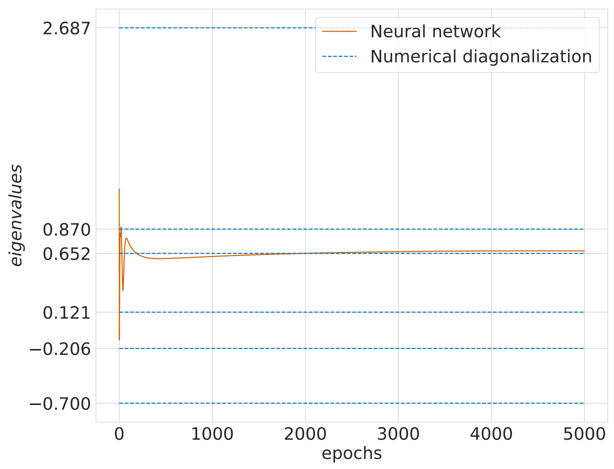


Figure 13: Estimate of the eigenvalue when initializing $x(0)$ to the eigenvector corresponding to the largest eigenvalue as a function of epochs using a neural network with a single hidden layer with 1000 nodes. The solution using `numpy`'s eigenvalue solver is shown as dashed lines.

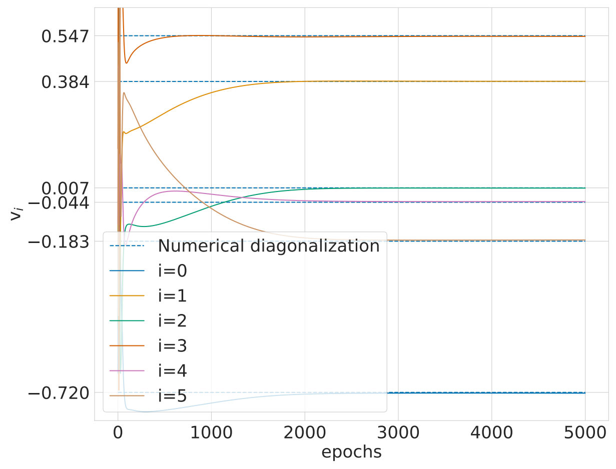


Figure 12: Estimate of the eigenvector components corresponding to the smallest eigenvalue as a function of epochs using a neural network with a single hidden layer with 1000 nodes. The solution using `numpy`'s eigenvalue solver is shown as dashed lines.

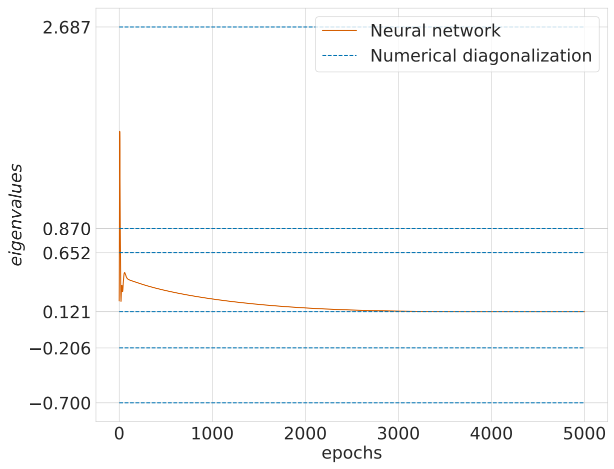


Figure 14: Estimate of the eigenvalue when initializing $x(0)$ to the eigenvector corresponding to the smallest eigenvalue as a function of epochs using a neural network with a single hidden layer with 1000 nodes. The solution using `numpy`'s eigenvalue solver is shown as dashed lines.

FTCS having a greater MSE at the beginning. The rate of change of the diffusion equation is greatly reduced after $t \approx 0.2$ and so the FTCS evolves toward a better and better fit as the rate of change diminishes. The FTCS method creates a solution as the method evolves in time and so we expect the MSE to be high where the PDE has greater oscillations. The FTCS method with $\Delta x = 0.01$ clearly reaches the lowest MSE, at $t = 0.714$ it reaches an MSE of magnitude 10^{-13} and keeps decreasing logarithmically.

The learning history of the neural network seen in Figure 5 shows a descending trend for the MSE as a function of epochs. The trend looks exponential with multiple regularly appearing spikes. The MSE starts to converge at about 5000 epochs which agrees with our original choice of an optimal number epochs, see

Appendix A Figure 15 for the learning history with 7000 epochs. Though we do not know the origin of the spikes we suspect that it's due to the ADAM optimiser. A simple check could have been done by switching out the optimiser but as optimisers aren't a topic in this report we will not study their impact on the neural network. The learning history using $\Delta x = 0.1$ and Δt dictated by the stability criteria for the FTCS method shows a high number of spikes in a concentrated area at about 3000 epochs. Again we do not know the reason for this, though we suspect that it's related to the asymmetry between the number points in time and space.

From Figure 7 it can seem that increasing the number of epochs does not improve the MSE. Though in this case it seems to be due to where in the cycle of spikes the learning of the neural network stopped, again see Appendix A Figure 15. Decreasing the MSE has no benefits or drawbacks that can be deduced from this image. The optimal number of epochs could have been better chosen, had we observed the learning history to begin with. For a trained neural network it seems setting $\Delta x = \Delta t$ has no benefits for the performance of the neural network as seen in Figure 8. The neural network acts as a continuous function as expected. The neural network clearly doesn't have to follow any stability criteria and the resolution in time and space can be chosen freely. We encountered a problem when training the model with step sizes $\Delta x = 0.01$ and $\Delta t = 5 \times 10^{-5}$ where the learning process would halt. With $\Delta x = 0.01$ and $\Delta t = 5 \times 10^{-5}$ there are 100 points in the spacial dimension and 2×10^4 points in time. Taking into consideration our 100 nodes per layer with two layers, the halting of the training could be due to the architecture not being able to handle such an amount of points. Increasing the resolution will increase the number of parameters to fit and so the architecture (layers and nodes per layer) might have to adapt to an increasing or decreasing complexity of points to fit. The MSE of the neural network's solution to the diffusion equation does not show an increasing or decreasing trend as t increases in Figure 7. The neural network is then less sensitive to oscillations in the solution of the PDE than the FTCS method.

Increasing the resolution in the spacial dimension of the FTCS method from $\Delta x = 0.1$ to $\Delta x = 0.01$ increased the computation time by a magnitude 10^3. Compared to the reduction in MSE it's a profitable increase. We tried decreasing the step size further to $\Delta x = 0.001$ but the method became computationally too expensive and we were not able to produce results. The neural network requires approximately 230.613s/epochs to train when using $\Delta x = 0.1$ and $\Delta t = 5 \times 10^{-3}$. However, once trained, the time required to produce solutions with different resolutions in time and space is reduced to about 2.2s. The neural network then requires more time to train before solving the diffusion equation than the FTCS method. A trained neural network produced results with varying resolutions in time and space faster than the FTCS method using spacial step size $\Delta x = 0.01$, approximately 0.4s faster.

The FTCS method is quick to implement, does not require tuning of parameters and produces results quickly with an acceptable MSE. The method is limited by its stability criteria and discretization, and is sensible to oscillations in the solution. It could be adapted to remove this criteria dependency for greater applicability, i.e. the Crank-Nicolson method [7]. Increasing the dimensionality of the problem would require a slightly different derivation for the evolution of f and possibly increase the computational cost of solving the PDE. The neural network does not rely on a stability criteria as it produces a function as a solution, which can be the reason why it is less sensible to oscillations in the PDE than the FTCS method. Solutions for specific timestamps can be computed by the neural network independent of previous and later timestamps once the training is done. The flexibility of the neural network is remarkable and shows that it can more easily be adapted to other PDEs with higher dimensionality. Increasing dimensionality and resolution of time and space would probably also come at a higher computational cost though we expect the implementation to be simple and less time consuming than increasing the dimensionality of the FTCS method. In the case of one-dimensional diffusion equation, both methods were here satisfactory.

B. Eigenvalue problem

We see in Figures 9, 11, 10 and 12 that the neural network is able to successfully converge to both the largest and smallest eigenvalues, as well as their corresponding eigenvectors. This is true when the initial vector is drawn from a normal distribution. These results are in accordance with the theory, which states that starting from any nonzero points, the solution will converge to an eigenvector of A. On the other hand, when starting from the eigenvector with the smallest or largest eigenvalue, we see in Figures 14 and 13 that we are not able to replicate the results in [1]. These results show that we are indeed able to converge to an eigenvalue of A, albeit not the corresponding eigenvalue of our starting vector. Given that we start from a vector that lies in the eigenspace of the extremal eigenvalues, the solution should be guaranteed to converge to the corresponding eigenvalue and eigenvector. A possible explanation could be that we get stuck at some local minima of the cost function. We found that a single hidden layer with large amounts of nodes yielded better results than multiple layers with fewer nodes. Due to limited hardware and not being able to employ GPU acceleration, we were not able to investigate a combination of a large number of nodes and several hidden layers. It should also be mentioned that considerable experimentation with random seeds had to be done in order to get the desired results. For future studies, we could look into different optimizers in combination with other activation functions and architectures.

V. CONCLUSION

We have applied the forward time-centered space method and a neural network with learning rate $\eta = 0.01$, 5000 epochs and the ADAM optimisation method to solve the one-dimensional diffusion equation. We found that the FTCS method with step sizes $\Delta x = 0.01$ and $\Delta t = 5 \times 10^{-5}$ gave the best results with a total computation time of 2.609s. However it had a limited applicability due to its stability criteria, discretization and sensitivity to oscillations of the solutions. All these weaknesses are addressed by the neural network which produces a function as a solution once trained. Both methods' performance were acceptable though for cases with higher dimensionality a neural network should be favoured.

We found that the MSE of the neural network's solution is heavily influenced by spikes in the learning history. Though their origin is unknown, we attribute them to the optimisation method ADAM.

We applied a neural network with a learning rate $\eta = 0.001$, 5000 epochs and the ADAM optimisation method to solve the eigenvalue problem as an ordinary differential equation. We found that the neural network was highly applicable to solve the problem when compared to a standard solver, with relative errors as low as $1.2 \times 10^{-4}\%$. The neural network did not converge to the lowest eigenvector and eigenvalue when starting in the corresponding eigenspace which we attribute to the neural network getting stuck during the learning process. As a side note, the reproduction of the author's work in [1] is challenging due to the lack of specifics around their implementation.

Altogether we have shown the wide applicability of neural networks for solving ordinary and partial differential equations and consequently eigenvalue problems. The flexibility of neural networks makes them favourable over finite difference methods.

[1] Z. Yi, Y. Fu, and H. J. Tang, Computers & Mathematics with Applications **47**, 1155 (2004).
[2] G. Cybenko, Mathematics of Control, Signals and Systems **2** (1989), 10.1007/BF02551274.
[3] A. Fick, Journal of Membrane Science **100**, 33 (1995).
[4] J. Charney, R. Fjörtoft, and J. Neumann, Tellus A **2** (1950), 10.3402/tellusa.v2i4.8607.
[5] N. T. A. Jakobsen, D. Haas and V. Ung, "Exploring the transition from regression models to neural networks," (2022).
[6] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, Journal of Machine Learning Research **18**, 1 (2018).
[7] J. Crank and P. Nicolson, Mathematical Proceedings of the Cambridge Philosophical Society **43**, 50–67 (1947).

Appendix A: Solving the PDE with a neural network and FTCS

We include the learning history of the neural network with 7000 epochs and the scaled comparisons at time $t = 0.8$ for the other mentioned solutions in section III.

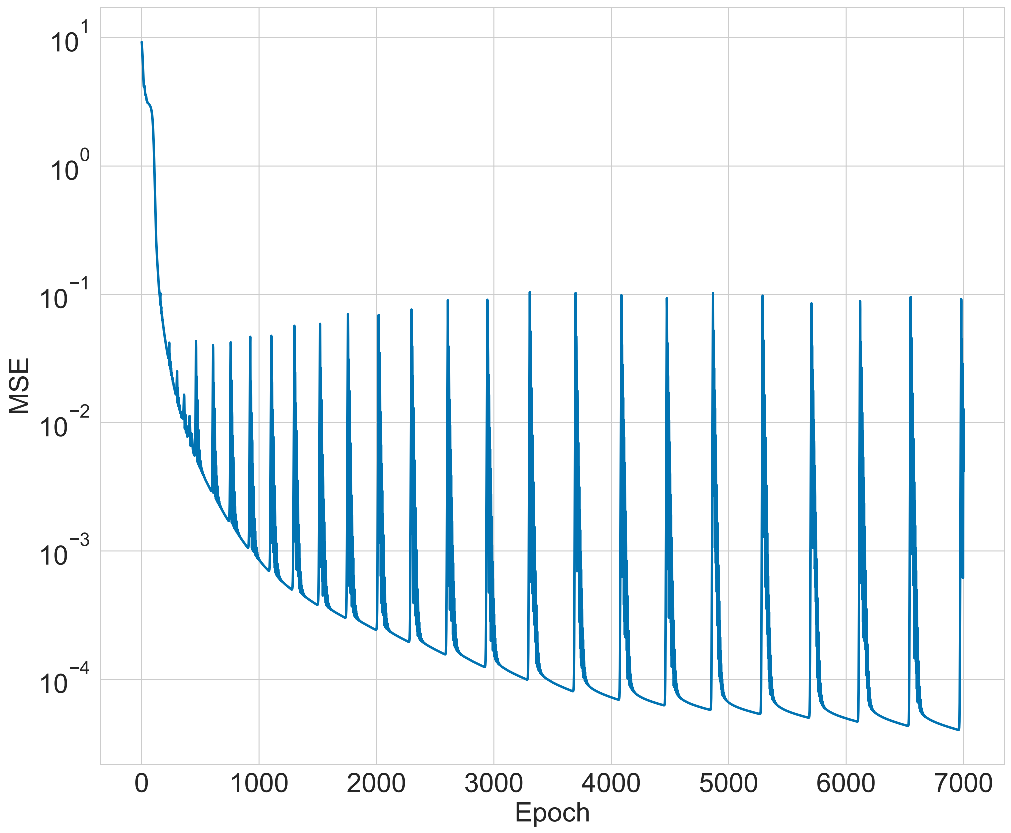


Figure 15: Learning history of the neural network with step sizes $\Delta x = 0.1$ and $\Delta t = 5 \times 10^{-3}$ and 7000 epochs. We used the ADAM optimiser and a learning rate $\eta = 0.01$.

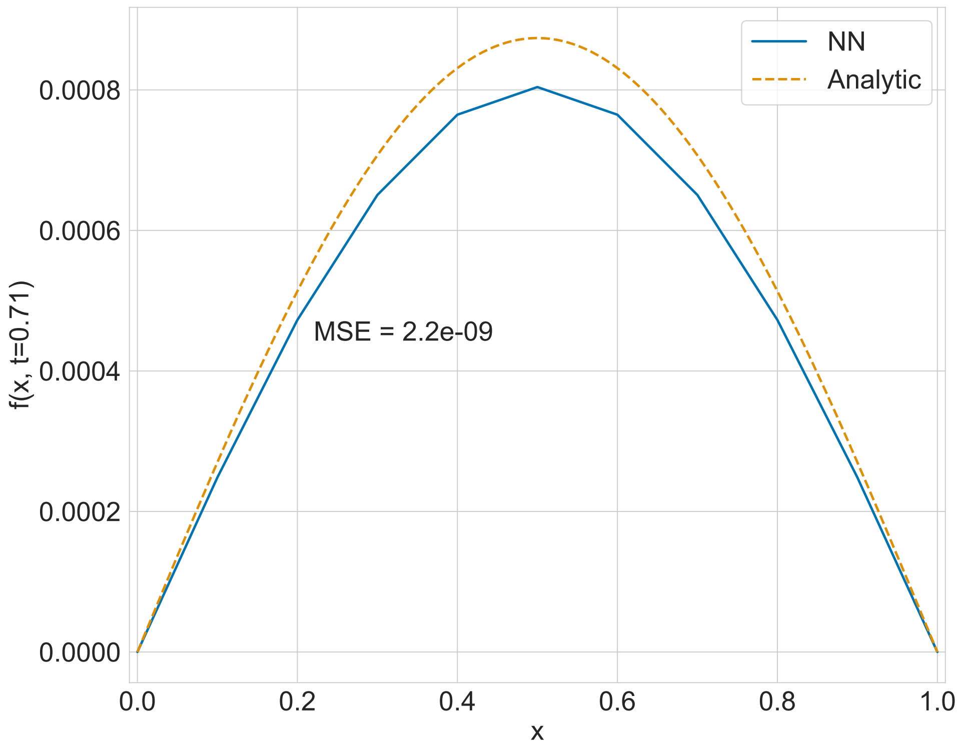


Figure 16: Comparison of the analytic solution of the diffusion equation to the FTCS solution solved using a step size in the space dimension $\Delta x = 0.1$. This comparison is at a time $t = 0.8$ with total time $T = 1$.

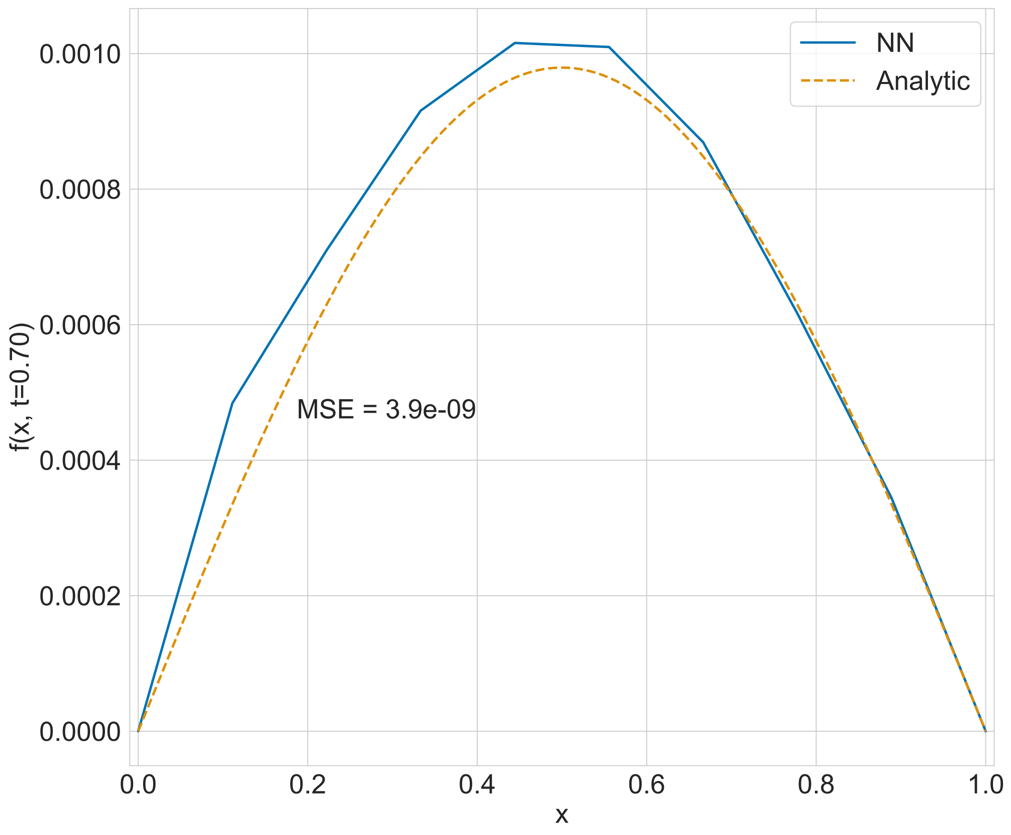


Figure 17: Comparison of the analytic solution of the diffusion equation to the neural network solution solved using step sizes $\Delta x = 0.1$ and $\Delta t = 5 \times 10^{-3}$. This comparison is at a time $t = 0.8$ with total time $T = 1$.