Solving a diffusion equation using neural networks

Sakarias Frette, William Hirst, Mikkel Metzsch Jensen

(Dated: December 1, 2021)

CONTENTS

I. INTRODUCTION

Differential equations are a cornerstone in understanding the physical processes around us. From the Schrödinger equation, which explains low energy quantum mechanical systems, to the Navier-Stokes equations that governs flow of fluids, they explain how the system impacts itself through change with respect to time or space or both. Traditional numerical solutions such as finite difference[3] and finite element[4] have yielded good results, and they are computationally fast, but they lack in one aspect. They base themselves on discretization, and so the solution you get is also discretized. Thus, you do not get a function with which can be evaluated at any given point. with the Neural ordinary differential equation paper[1] from 2018 and other papers, it was shown that neural networks also can solve both ODE's and PDE's. With these networks one can get a actual function that can be evaluated, though at a potential cost of computation time.

It was also shown in 2003[5] that neural networks can be used to solve eigenvalue problems. Eigenvalue problems occur in many physical phenomenon, such as the Schrödinger equation, where the energy of the system is the eigenvalue of the Hamilton operator, or in vibration analysis, where the eigenvalue corresponds to the eigen-frequency.

This is the motivation for this report, where we will use both finite difference and machine learning to solve the diffusion equation and compare computation time and error, as well as compute eigenvalues for a given matrix using machine learning.

II. THEORY

A. The diffusion equation

The diffusion equation in 1D in it's simplest form is given as

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

and in our case is defined in the domain of

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

for $t > 0$, with the following initial condition

$$u(x,0) = \sin(\pi x)\ ;\ 0 < x < L,$$

and with Zero-Dirichlet boundary conditions

$$u(0,t) = u(L,t) = 0\ ;\ t \geq 0.$$

1. *Analytical solution*

This equation has an analytical solution, which we will use to test our network's solution. This is found by analysing the equation and "guessing" on a general form. From our equation we see that we need a function that is its own second derivative, namely a sine or cosine function. We see also that our initial condition is a sine function, which supports that guess. We also need a function that does not change when using a time derivative operator on it, and a exponential function is thus a good guess. we then have

$$u_e(x,t) = A\sin(kx)e^{-\omega t}$$

Here we have three degrees of freedom, A, k and ω. We first find ω by putting our guess function into equation 1:

$$\frac{\partial^2 u_e(x,t)}{\partial x^2} = -k^2 A \sin kx e^{-\omega t},$$

$$\frac{\partial u_e(x,t)}{\partial t} = -\omega A \sin(kx)e^{-\omega t}.$$

This relation gives us

$$\omega = k^2.$$

Next we implement the Zero-Dirichlet boundary condition:

$$A\sin(k\cdot 0)e^{-k^2 t} = A\sin(kL)e^{-k^2 t} = 0,$$

$$\sin(kL) = \sin(k\cdot 0),$$

$$kL = \arcsin 0 = N\pi,$$

$$k = \frac{N\pi}{L}.$$

Finally we implement the initial condition

$$A\sin(\frac{\pi}{L}x)e^{(\frac{\pi}{L})^2\cdot 0} = \sin(\pi x),$$

where we choose the length of the rod L to be 1, so we get that

$$A\sin(\pi x) = \sin(\pi x).$$

Thus $A = 1$. Our final, analytical solution is then

$$u_e(x,t) = \sin(\pi x)e^{-(\pi)^2 t}. \tag{2}$$

2. *Finite difference discretization*

To solve this equation with a computer we need to use numerical discretization. One such method is finite difference, where we use approximations to the derivatives. The simplest case here is to use the so-called explicit Forward Euler method, which uses forward difference on first order derivatives, and center difference on second order derivatives.

By convention we will use subscript indexing for spacial coordinates and superscript indexing for temporal coordinates, where j is the index to the numerical solution for time, and i is the index for spacial coordinate. This convention is somewhat arbitrary, but helps if one extends the diffusion equation to more dimensions in space. We will also use the convention that $u(x_i, t_j) = u_i^j$, where x_i is the ith element in space, and t_j is the jth element in time.

Thus, the first order derivative with respect to time in equation 1 is given as

$$\frac{\partial u(x,t)}{\partial t} \approx \frac{u_i^{j+1} - u_i^j}{\Delta t}.$$

The second order derivative with respect to space uses center difference, and is given as

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2}.$$

Assembling these two discretizations together we get that

$$\frac{u_i^{j+1} - u_i^j}{\Delta t} = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2}.$$

$$u_i^{j+1} = u_i^j(1 - \frac{2\Delta t}{\Delta x^2}) + \frac{\Delta t}{\Delta x^2}(u_{i+1}^j + u_{i-1}^j). \tag{3}$$

This equation has a stability criteria which must be satisfied for the numerical solution. The general criteria is given as

$$\frac{\alpha \Delta t}{\Delta x^2} \leq \frac{1}{2},$$

where α is the constant multiplied with the spacial part of the diffusion equation. In our case $\alpha = 1$, which allows for reasonably small step sizes, but for large α this method requires too small step sizes for practical use.

B. Neural networks and PDE's

1. *Trial function*

To solve this problem we need to contruct a trial function, $g_t(x,t) \approx u(x,t)$, defined as

$$g_t(x,t) = h_1(x,t) + h_2(x,t)N(x,t,P), \tag{4}$$

where $N(x,t,P)$ is the data from the neural network, $h_1(x,t), h_2(x,t)$ are functions that ensures that the boundary and initial conditions are kept. We have zero-Dirichlet conditions on the boundaries, so a function like $x(1-x)$ ensures that the function is zero at the boundaries. We also have an initial condition, so the final trial function is thus

$$\begin{aligned} g_t(x,t) &= h_1(x,t) + h_2(x,t)N(x,t,P) \\ &= (1-t)\sin(\pi x) + x(1-x)tN(x,t,P) \end{aligned} \tag{5}$$

2. *Cost function*

When using supervised learning one usually has a well defined cost function testing the model against the exact data. But in the case of solving PDE's, it is not given that one has access to an exact solution to compare against. We thus introduce a different cost function. The cost function uses the residual of the differential equation to calculate the loss. The residual is defined as putting all the terms on one side of the equation, and in the case of the diffusion equation, we have that the residual R is

$$R = \frac{\partial u(x,t)}{\partial t} - \frac{\partial^2 u(x,t)}{\partial x^2} = 0. \tag{6}$$

We then square the residual and take the mean, giving us the cost function

$$C(x,t) = mean(R^2) = mean\left(\left[\frac{\partial u(x,t)}{\partial t} - \frac{\partial^2 u(x,t)}{\partial x^2}\right]^2\right). \tag{7}$$

Thus, our problem has become a minimization problem with respect to the calculation it self.

C. Finding the eigenvalues of a symmetric matrix through a neural network

III. IMPLEMENTATION

The code can be found on Github.

A. Explicit solver

The explicit solution was calculated using algorithm 1.

Algorithm 1: Simulate system using vectorized solution.

```
1  Initialize complete solution array, $u_{complete}$ ;
2  Initialize domain;
3  Initialize array for solution at t, u Initialize array
     for solution at $t - \Delta t$, $u_1$;
4  Insert initial condition ;
5  for $1 \rightarrow t_N$ do
6      Calculate the inner spacial points for u;
7      Insert boundary conditions;
8      Update u, $u_1$;
9      Insert solution to $u_{complete}$;
10 end
11 Return $u_{complete}$
```

Note here the use of three arrays, namely the final solution $u_{complete}$, u and u_1. Matrices are generally more computationally heavy to compute, and they take up large amounts of storage in the computer memory. Thus, using arrays to house the solution t and $t - \Delta t$ is more cost effective.

B. Tensorflow

To compare with the exlicit solution we used TensorFlow infrastructure to calculate the numerical solution of the PDE. The pseudo code used is listed in algorithm 2

Algorithm 2: Training of the network.

```
1 Initialize model ;
2 Initialize Total loss list;
3 for epoch in epochs do
4     Calculate the loss from model;
5     Calculate gradient and loss with respect to
        hidden variables;
6     Apply gradient to model;
7     Append loss to total loss;
8 end
9 Return total loss
```

1. *Optimizers*

The code uses the ADAM[2] optimizer. This method uses an adaptive momentum. Lets say that the algorithm gets stuck at a local minimum. The ADAM optimizer tries then an increase in the learning rate to see if the algorithm can "jump" out of the minimum.

2. *Data*

The data set here has two features, namely position, x, and time, t. We want the network to learn by computing the derivatives, and thus, we need every possible combination of the two features. To do this, we create a meshgrid.

3. *Structure of the network*

To solve the PDE and the eigenvalue problem, we structured the data as shown in table I

Table I: Each layer has a given set of nodes, with number of tunable parameters and an activation function

Layer	Nodes	Params	Activation
1	20	60	Sigmoid
2	20	420	Sigmoid
3	20	420	Sigmoid
4	1	21	None

This number of layers and nodes where not arbitrarily chosen. We tested for multiple combinations but found that this combination gave the best final loss.

IV. RESULTS AND DISCUSSION

A. Diffusion Equation

In this section we will by studying the diffusion equation discussed in section II A.

1. *Finite Difference*

The results from using finite difference discretization are shown in figures 1 and 2.

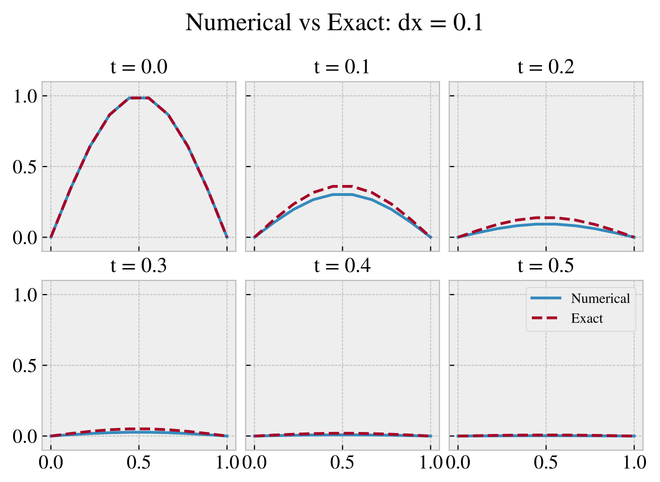

Figure 1: Explicit solution vs exact solution for diffusion equation with $dx = 0.1$, $dt = 5e^{-5}$, $L = 1$ and $T = 0.5$.

In figure 1 we time evolution of the numerical solution against the exact solution as function of space for several times t. We see here that the numerical solution gives lower values for u than the exact solution.

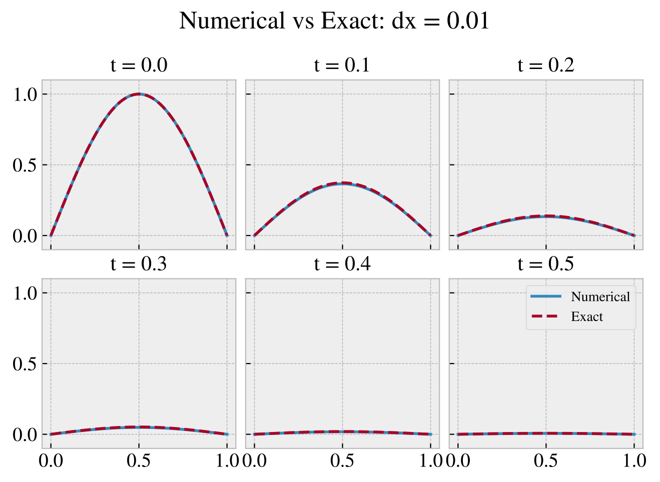

Figure 2: Explicit solution vs exact solution for diffusion equation with $dx = 0.01$.

In figure 2 we time evolution of the numerical solution against the exact solution as function of space for several times t. We see here that the numerical solution is almost identical to the exact solution.

2. *Neural Network*

The results from using the neural network to calculate the PDE solution is shown in figures (3) and (4).

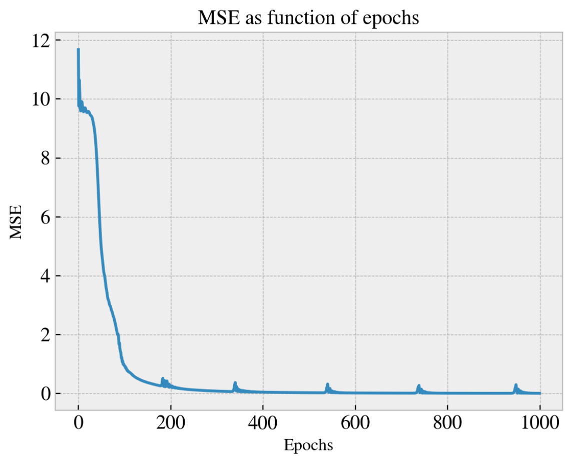

Figure 3: Tensorflow MSE as function of epochs, using $\eta = 0.05$, 1000 epochs, $dt = dx = 0.01$, $L = 1$ and $T = 1$.

In figure 3 we observe that the mean squared error falls down rapidly, and then stabilizes around 0.004. The sawtooth behavior is due to the ADAM optimizer, which tries to see if the solution is stuck in a local minimum, and not the global minimum. From the plot we can assume that it has found the global minimum.

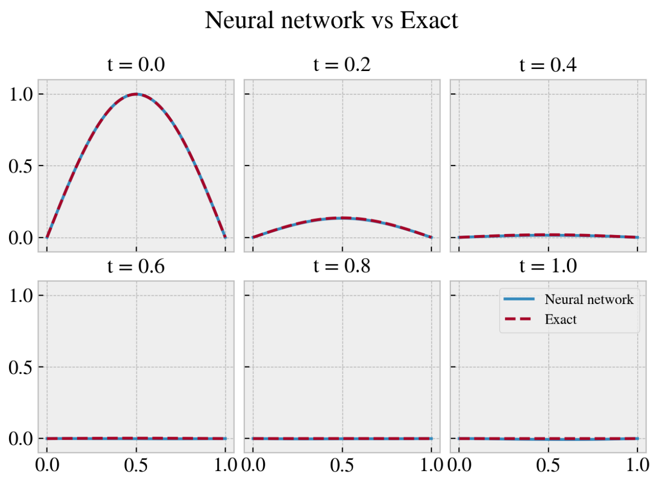

Figure 4: Tensorflow solution against exact solution for $t = 0, 0.2, 0.4, 0.6, 0.8, 1$, using $\eta = 0.05$, 1000 epochs, $dt = dx = 0.01$, $L = 1$ and $T = 1$.

In figure 4 we see the machine learning solution against the exact solution for $t = 0, 0.2, 0.4, 0.6, 0.8, 1$. Here we observe that the solution fits very well with the exact solution. A gif of the entire solution can be found in the article/animations folder in the Github project3 folder.

To compare the two methods, one can look at the mean relative error. Here we have solution for several time steps, so we calculate the relative error for one time step and then take the mean. Thus we get figure 5.

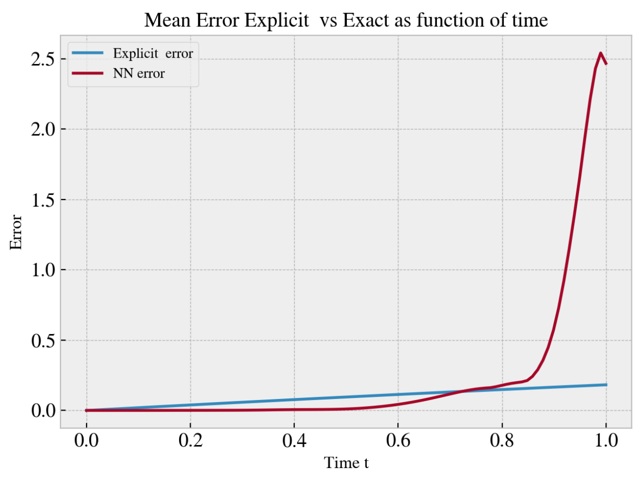

Figure 5: Tensorflow and explicit solution as function of time with $\eta = 0.05$, 10^5 epochs, $dt = dx = 0.01$ for the network, $dx = 0.01$ and $dt = \frac{dx^2}{2}$, $L = 1$ and $T = 1$.

In figure 5 we see the mean relative error as function of time for both the neural network and the explicit solution. We observe here that close to $t = 1$ the neural network has much higher error than the explicit solution. Even at 10^5 epochs it does worse, and at lower number of epochs, the relative error gets higher and higher.

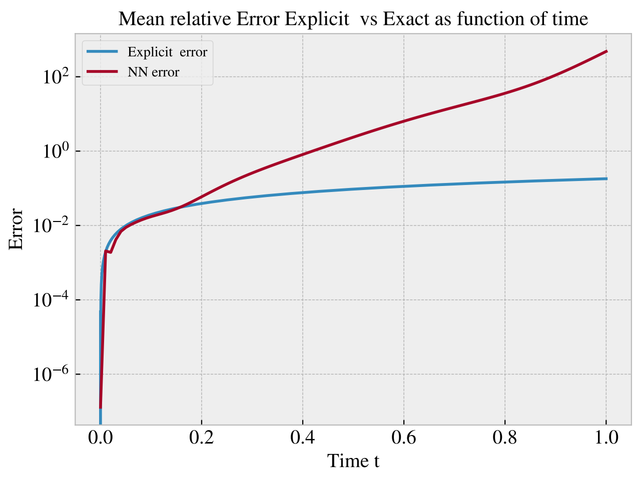

Figure 6: Log plot of Tensorflow and explicit solution as function of time with $\eta = 0.05$, 10^3 epochs, $dt = dx = 0.01$ for the network, $dx = 0.01$ and $dt = \frac{dx^2}{2}$, $L = 1$ and $T = 1$, using sigmoid as activation

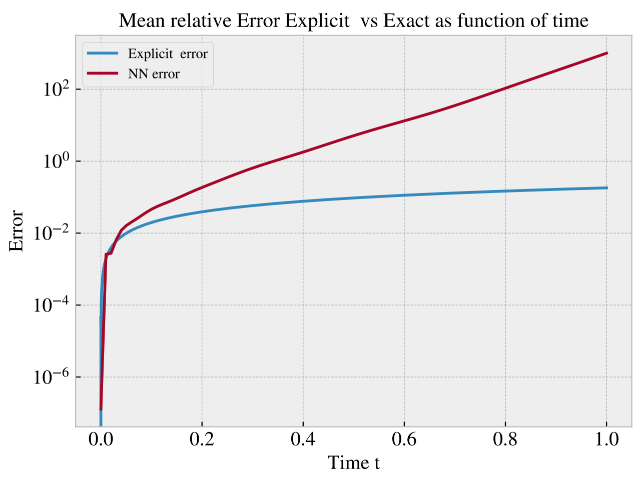

Figure 7: Log plot of Tensorflow and explicit solution as function of time with $\eta = 0.05$, 10^3 epochs, $dt = dx = 0.01$ for the network, $dx = 0.01$ and $dt = \frac{dx^2}{2}$, $L = 1$ and $T = 1$, using tanh as actication.

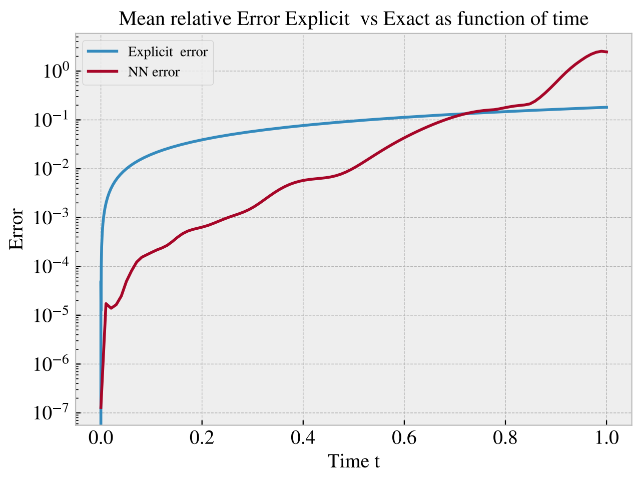

Figure 8: Log plot of Tensorflow and explicit solution as function of time with $\eta = 0.05$, 10^5 epochs, $dt = dx = 0.01$ for the network, $dx = 0.01$ and $dt = \frac{dx^2}{2}$, $L = 1$ and $T = 1$.

3 v 2 to explicit solution. explicit is faster, better relative error and easier to create code. Neural networks gives a function that can be evaluated everywhere, but will not be as accurate far away from training domain, and is not dependent on stability criteria.

B. Eigen value problem

In this section we have studied the the eigen value problem discussed in section II C.

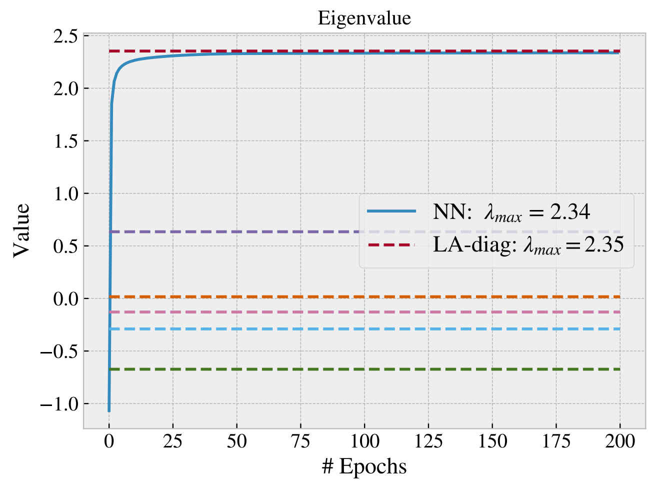

Figure 9: Eigen values for a 6x6 matrix.

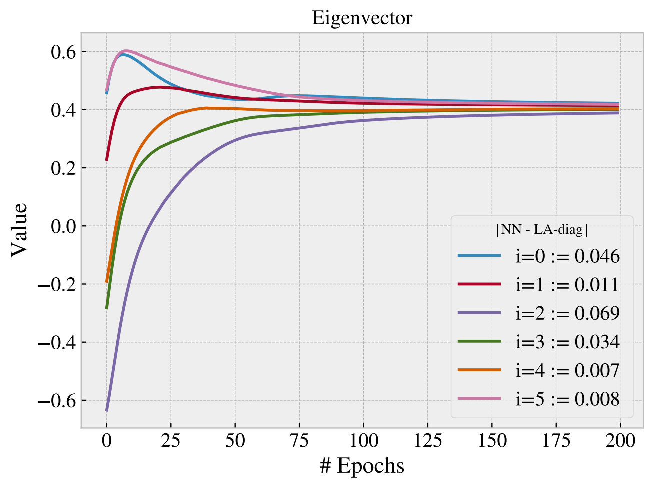

Figure 10: Eigenvectors for a 6x6 matrix.

The corresponding eigenvector.

V. CONCLUSION

Hmm, something something

REFERENCES

[1] David Deuvenad et al. “Neural Ordinary Differential Equations.” In: (2018). DOI: https://arxiv.org/pdf/1806.07366.pdf.

[2] Diederik Kingma and Jimmy Lei Ba. “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION.” In: (2015). DOI: https://arxiv.org/pdf/1412.6980.pdf.

[3] Hans Petter Langtangen. *Finite Difference Computing with PDEs*. Texts in Computational Science and Engineering. Springer, 2016. ISBN: 978-3-319-55455-6.

[4] Hans Petter Langtangen. *Introduction to Numerical Methods for Variational Problems*. Texts in Computational Science and Engineering. Springer, 2016. ISBN: 978-3-319-55455-6.

[5] Zhang Yi, Yan Fu, and Hua Jin Tang. “Neural Networks Based Approach Computing Eigenvectors and Eigenvalues of Symmetric Matrix .” In: (2003). DOI: https://www.sciencedirect.com/science/article/pii/S0898122104901101.