FYS-STK4155 - Project 3

Solving Partial Differential Equations Using Artificial Neural Networks

Eirill Strand Hauge – Anders Julton

(Dated: December 18, 2019)

The goal of this project was to solve differential equations using artificial neural networks. We solved the partial differential equation *the heat flow equation* using a neural network and compared the performance to that of the numerical implementation of a finite difference method. For the example equation used, *temperature gradient in a rod*, the mean squared error when using the finite difference method was $3 \cdot 10^{-6}$ when $\Delta x = 0.1$ and $3 \cdot 10^{-10}$ when $\Delta x = 0.1$, both using $\Delta t = \Delta x^2/2$. The artificial neural network successfully solved the partial differential equation, but was outperformed by the finite difference method with the smaller step length. The artificial neural network using two hidden sigmoid-layers and a linear output layer scored $\sim 10^{-9}$ after 20000 back propagations.

We also extended the neural network to compute the eigenvalues of randomly generated real symmetric 6×6 matrices. While the eigenvalue the network converged to was random, multiple runs of the same matrix through the network eventually yielded all the 6 eigenvalues However the runtime of the network, 364 seconds in one example, far exceeded that of an eigenvalue solver from a standard and public library, which had a runtime of less than one second.

I. INTRODUCTION

In this project we will attempt to solve the heat equation, a partial differential equation (PDE) that describes the time evolution of the distribution of temperature in a solid medium. Two methods for solving the PDE will be implemented, the explicit Forward Euler numerical method and an artificial neural network. The example problem that will be studied is the one dimensional temperature gradient in a rod, with boundary and initial conditions.

Furthermore we will attempt to compute eigenvalues of randomly generated 6×6 matrices using the same neural network used for the heat equation, where the eigenvalue problem will be formulated as an ordinary differential equation. The modified neural network will be compared to an eigenvalue solver from a standard and public linear algebra package.

While we believe the neural network will be able to solve the differential equations, we are skeptical as to how well it will performed compared to the numerical solution for the heat flow equation, and the eigenvalue solver for the eigenvalue problem. Specifically we believe the network will be outmatched when comparing run times.

We will start by introducing the differential equations and how we intend to solve them. We will then briefly describe the choices made for the implementation. The results will then be presented before discussing their significance. Lastly, a conclusion will be presented. Each section will be somewhat divided into two parts. The first part will cover the partial differential equation problem, the heat wave equation, while the second part will cover the ordinary differential equation used to solve the eigenvalue problem.

II. METHOD

A. The Heat Flow Equation

The heat flow equation in one dimension is

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

a partial differential equation where $u(x,t)$ is the temperature at position $x \in [0, L]$ at time $t \geq 0$. The parameter c^2 is a constant describing the thermal diffusivity of the material.

To solve any such PDE, the initial condition $f(x) = u(x,0)$ and the boundary conditions must be known. We will limit our discussion to the boundary conditions

$$u(0,t) = u(L,t) = 0. \tag{2}$$

This equation is solved analytically by using separation of variables. When using separation of variables, we impose that the function $u(x,t)$ can be written as the product of two single-variable functions $F(x)$ and $G(t)$, that is $u(x,t) = F(x)G(t)$, giving the new expression for the partial differential equation

$$G(t)\frac{d^2F(x)}{dx^2} = c^2 F(x)\frac{dG(t)}{dt}. \tag{3}$$

For a more compact notation, we rename the derivatives as $\dot{G}(t) \equiv \frac{dG(t)}{dt}$ and $F''(x) \equiv \frac{d^2F(x)}{dx^2}$.

Rewriting the heat flow equation, we can get each side of the equation to depend solely on one single variable, as shown below

$$\frac{F''(x)}{F(x)} = c^2 \frac{\dot{G}(t)}{G(t)}. \tag{4}$$

Now, the important observation from this equation, is that both $\frac{F''(x)}{F(x)}$ and $\frac{\dot{G}(t)}{G(t)}$ must equal a constant, denoted $-k^2$ in the derivations below, as the equation must hold for all combinations $x \in [0, L]$ and $t \geq 0$. From this we get the two equations

$$\frac{F''(x)}{F(x)} = -k^2 \tag{5}$$

$$\frac{\dot{G}(t)}{G(t)} = -\frac{k^2}{c^2}, \tag{6}$$

from which we can see that there are infinite solutions fulfilling the boundary conditions on the form

$$F_n(x) = a_n \sin(k_n x) \tag{7}$$

$$G_n(x) = e^{-\lambda_n^2 t}, \tag{8}$$

with $k_n = \frac{n\pi}{L}$, $\lambda_n = \frac{k_n}{c}$ and n can be any positive integer, excluding the trivial case $n = 0$. This gives the set of eigenfunctions

$$u_n(x,t) = a_n \sin(k_n x) e^{-\lambda_n^2 t}. \tag{9}$$

The general solution $u(x,t)$ is a linear combination of the eigenfunctions $u_n(x,t)$,

$$u(x,t) = \sum_{n=1}^{\infty} a_n \sin(k_n x) e^{-\lambda_n^2 t}. \tag{10}$$

The coefficients a_n are decided by the initial condition $u(x,0)$ by

$$a_n = \frac{2}{L} \int_0^L u(x,0) \sin(k_n x) dx. \tag{11}$$

We have now seen the general analytic solution to this problem. A simple example will now be introduced, which will be the example used when later testing the implementations.

1. Temperature Gradient in a Rod

We give an example of the heat flow equation which is the temperature gradient in a rod. We have a rod where both ends of the rod are kept cool. To simplify, the length of the rod is set to $L = 1$. This gives the boundary conditions $u(0,t) = u(1,t) = 0$.

At time $t = 0$ the rod is warm in the middle, and gradually cooler as the we approach the edges. The initial condition is given by

$$u(x,0) = \sin \pi x, \tag{12}$$

from which we can find the Fourier constants by

$$a_n = 2 \int_0^1 \sin(\pi x) \sin(k_n x) dx \tag{13}$$

which is zero for all n except $n = 1$ which gives $a_1 = 1$. This gives us the full solution

$$u(x,t) = \sin(\pi x) e^{-\pi^2 t}, \tag{14}$$

setting the constant in equation (1) to $c^2 = 1$.

Before moving on the different methods for approximating the heat wave equation, we will introduce the equations that will be used to asses the methods.

2. Assessing Approximated Solutions

In the two upcoming sections, two different approaches to solving the heat flow equation will be introduced. To assess the results, three measurements will be used. First, we have the mean absolute error at time t summed over a set of x-values

$$\bar{E}(t) = \sum_i |u(x_i,t) - \tilde{u}(x_i,t)|, \tag{15}$$

where $u(x,t)$ is the analytical solution and $\tilde{u}(x,t)$ is the computed solution using neural networks or the finite difference method.

We then have the mean relative (and absolute) error at time t given by

$$\bar{E}_{rel}(t) = \sum_i \frac{|u(x_i,t) - \tilde{u}(x_i,t)|}{u(x_i,t)}. \tag{16}$$

Lastly, the mean squared error over time and space will be used. This is given by

$$MSE = \sum_n \sum_i \left(u(x_i,t_n) - \tilde{u}(x_i,t_n)\right)^2. \tag{17}$$

We will now introduce a classical numerical method for solving the heat flow equation.

B. Numerical Solution to the Heat Flow Equation

The differential equation in eq. (1) will be solved numerically using centered difference in space and forward difference in time. The compact operator notation for finite differences used will be D_x for centered difference and D_t^+ for forward difference. For each time $t_n = n\Delta t$ and position $x_i = i\Delta x$, the numerical solution will be according to

$$\left[D_x D_x u = c^2 D_t^+ u\right]_i^n, \tag{18}$$

where the steps lengths Δx and Δt are held constant.

A general formula for discretizing the problem given by eq. (18) is needed for the numerical implementation. In order to get a better overview, the problem is divided into two, corresponding to the left and the right hand side. We will begin with the left hand side of the equation.

First we take the double derivative with respect to x using centered finite difference

$$\begin{aligned}[D_xD_xu]_i^n = [D_x(D_xu)]_i^n &= \frac{[D_xu]_{i+\frac{1}{2}}^n - [D_xu]_{i-\frac{1}{2}}^n}{\Delta x} \\ &= \frac{u_{i+1}^n - u_i^n}{\Delta x^2} - \frac{u_i^n - u_{i-1}^n}{\Delta x^2} \\ &= \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}.\end{aligned} \tag{19}$$

The right hand side of the equation has a derivative with respect to t. The numerical expression using forward difference is

$$c^2[D_t^+u]_i^n = \frac{u_i^{n+1} - u_i^n}{\Delta t}c^2. \tag{20}$$

Inserting these results into eq. (18) yields

$$\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} = \frac{u_i^{n+1} - u^n}{\Delta t}c^2, \tag{21}$$

which gives the final iterative scheme

$$u_i^{n+1} = u_i^n + \frac{\Delta t}{\Delta x^2 c^2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right). \tag{22}$$

The solution requires an initial condition u_i^0 and set boundary conditions.

Next we will see how the same partial differential equation can be solved using an artificial neural network.

C. Solving The Heat Flow Equation Using Artificial Neural Networks

The heat flow equation in eq. (1) can be solved using artificial neural networks. We will be using feed forward layers. The network must be provided a trial function, $u_{NN}(x,t)$, which the neural network optimize such that the trial function approximates the solution to the heat equation, $u_{NN}(x,t) \approx u(x,t)$. The trial function must ensure that the boundary conditions are fulfilled. The trial function used in this project is given by

$$u_{NN}(x,t) = u_{t=0}(x) + x(L-x)ta^{out}(x,t), \tag{23}$$

where $u_{t=0}(x)$ is the exact initial condition of the system and $a^{out}(x,t)$ is the activation value of the output layer. The part $x(L-x)$ multiplied by the activation value ensures the boundary conditions being fulfilled, while multiplying the activation value by t ensures the boundary condition.

The Artificial Neural Network is trained by using a cost function $C(\boldsymbol{\theta})$, where θ symbolizes all trainable variables of the neural network (weights between neurons and bias of each neuron). The cost function should evaluate how good a solution $u_{NN}(x,t)$ is to the partial differential equation (1). The cost function chosen in this the mean squared error

$$C(\boldsymbol{\theta}) = \sum_k \left(\frac{\partial^2 u_{NN}(x_k,t_k)}{\partial x^2} - c^2\frac{\partial u_{NN}(x_k,t_k)}{\partial t}\right)^2, \tag{24}$$

summing over all combinations of time and position (x_k, t_k) of a given data set. The back propagation of the neural network will minimize the cost function with respect to the trainable variables, $\boldsymbol{\theta}$, using the ADAM optimizer.

Moving from a partial differential equation to an ordinary differential equation, we will use much of the same concepts as just presented. We will now explain how to define an eigenvalue problem as a ordinary differential equation.

D. Solving Eigenvalue Problems Using Artificial Neural Networks

We will aim to solve the eigenvalue problem $\mathbf{Av} = \lambda\mathbf{v}$ using artificial neural networks. We will labbel the eigenvalues and corresponding eigenvectors such that $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$.

Given a real symmetric $n \times n$ matrix $\mathbf{A}$, Yi *et al.* showed in their article [4] that the largest eigenvalue of A can be found by using a recurrent neural network (RNN) governed by

$$\frac{d\mathbf{x}(t)}{dt} = -\mathbf{x}(t) + f(\mathbf{x}(t)), \text{ for } t \geq 0, \tag{25}$$

where $\mathbf{x}(t)$, a vector of length n, is the output from the artificial neural network given a value t as input, and

$$f(\mathbf{x}(t)) = [\mathbf{x}^T(t)\mathbf{x}(t)\mathbf{A} + (1 - \mathbf{x}^T(t)\mathbf{A}\mathbf{x}(t)\mathbf{I}]\mathbf{x}(t),$$

and $\mathbf{I}$ is the $n \times n$ identity matrix. The smallest eigenvalue is found by replacing $\mathbf{A}$ with $-\mathbf{A}$. We see that if the right hand side of the equation (25) is zero, $\mathbf{x}(t)$ must be an eigenvector or a linear combination of eigenvector of $\mathbf{A}$.

From this it is easy to see that the system reaches equilibrium if $\mathbf{x}(t)$ evolves into an eigenvector of $\mathbf{A}$. Yi *et al.* showed that this will happen in the limit $t \to \infty$, for any starting point $\mathbf{x}(0)$.

In this project we will not use a RNN to evolve the state $\mathbf{x}(t)$ into an equilibrium state, but rather use the same network that will be used solve the heat flow equation, explained in the previous section, to find the equilibrium state directly. The cost function will again be

the mean squared error

$$C(\boldsymbol{\theta}) = \left(\frac{d\mathbf{x}(t)}{dt} + \mathbf{x}(t) - f(\mathbf{x}(t)) \right)^2. \tag{26}$$

The back propagation of the neural network will minimize the cost function with respect to the trainable variables, $\boldsymbol{\theta}$, using the ADAM optimizer.

This way, we will find an approximation to an eigenvector $\mathbf{x}(t) \approx \mathbf{v}_i$. The eigenvalue will then be found using

$$\lambda_i = \frac{\mathbf{x}^T(t)\mathbf{A}\mathbf{x}(t)}{\mathbf{x}^T(t)\mathbf{x}(t)}, \tag{27}$$

where the predictions of Yi is that in the limit $t \to \infty$, the artificial neural network should return the largest eigenvalue, λ_n.

Next section will cover a short description of the implementation of these methods.

III. IMPLEMENTATION

The implementation language chosen was `Python`. All figures used shown in the results section were procured using the `matplotlib` library. The full implementation can be found at `https://github.com/eirillsh/NeuralNetwork_PDE`.

A. Analytical Solution the Heat Flow Equation

The class `HeatFlow` represents the eigenfunctions of the heat flow equation with zero boundaries, as shown in eq. (9). The implementation was tested using `autograd` [2] to ensure that the eigenfunctions fulfilled the heat flow equation (1).

B. Finite Difference: Numerical Solution the Heat Flow Equation

The class `FiniteDifference` creates instances corresponding to the heat flow equation (1) given boundaries and the initial condition. The class solves the heat flow equation numerically using an iterative scheme as defined in eq. (22). The iterative scheme was vectorized in each time step using the `NumPy` library [3]. The implementation was tested by comparing calculations to the exact solution from `HeatFlow`, using small step lengths and propagating over a short time-period in order to minimize the numerical error.

C. Artificial Neural Neural Network for Solving the Heat Wave Equation

An abstract class constructing an artificial neural network called `NeuralNetwork` was implemented building on the `tensorflow` library [1]. The class `NeuralNetwork` inherits from the model `Sequential` from `tensorflow`'s `keras`. The neural network was built on feed-forward layers from the `keras` library. For the back propagation, or the process of optimizing the network, the `ADAM` optimizer from the `keras` library was used. The class uses an abstract method for the cost function.

The neural network implemented for solving the heat wave equation, `HeatFlowSolver`, inherits from the class `NeuralNetwork`. The method for the cost function was overwritten using eq. (24).

D. Artificial Neural Neural Network for Solving Eigenvalue Problems of a Symmetrical Matrix

The neural network implemented for solving the eigenvalue problem, `EigenProblem`, inherits from the class `NeuralNetwork`. The method for the cost function was overwritten using eq. (26).

IV. RESULTS

A. The Heat Flow Equation

Two different methods for solving the heat flow equation (1) was tested in this project. Both studies aimed to solve the problem of a temperature gradient in a rod, a problem described in the method section.

Two points in time where chosen to visualize and compare the performance of the two different methods. Below, in figure (1), the analytical solution to the chosen heat flow equation, as given by eq. (14) is shown for the initial condition and the two chosen points in time.

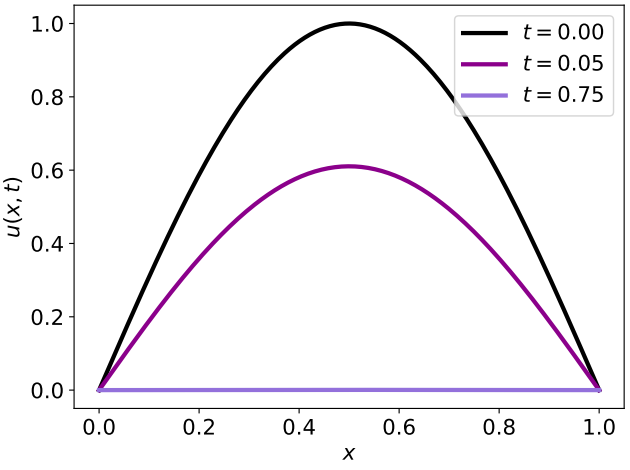

FIG. 1: Analytical solution at selected points in time to the heat flow equation called temperature in a gradient rod, as given by eq. (14).

The first point in time, $t = 0.05$, was chosen soon after

the initial condition such that the line should still be curved. The other time, $t = 0.75$, was chosen close to the equilibrium state.

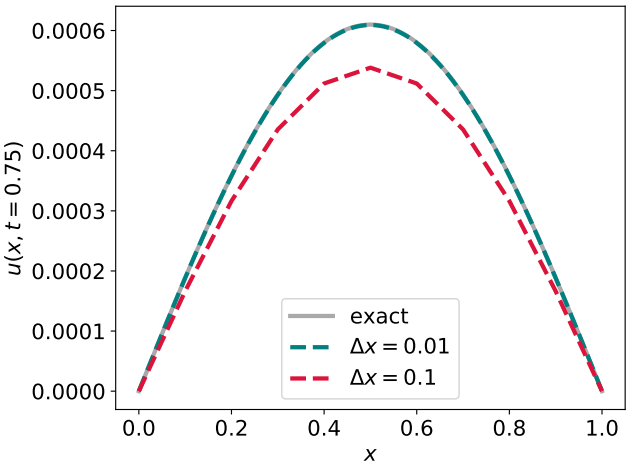

FIG. 3: Solution to the temperature gradient in a rod at time $t = 0.75$ using the numerical method of finite difference. For both Δx, the corresponding Δt was set to $\Delta t = \Delta x^2/2$. The gray line marked "exact" shows the analytical solution.

1. *Heat Flow Equation by Finite Difference*

The performance of the finite difference scheme as shown in eq. (22) was studied using the example of the temperature gradient in a rod. The results can be found in this section.

The figures below (2 - 3) show the calculated solution to the temperature gradient in a rod at the two chosen points in time ($t = 0.05$ and $t = 0.75$), using finite difference.

A study of the error of the finite difference compared to the analytical solution was made. The error is shown as a function of time on the figures below (4 - 5).

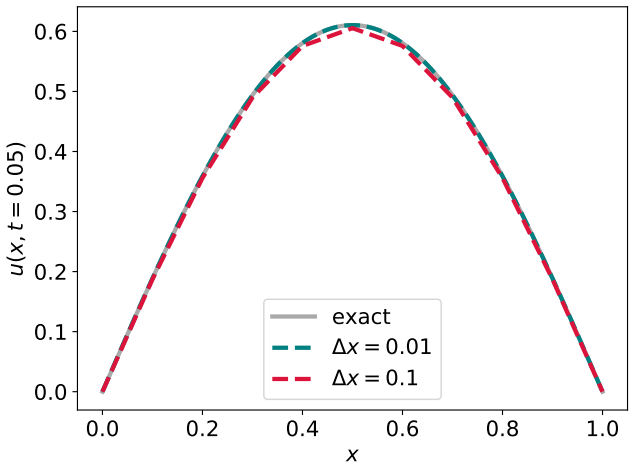

FIG. 2: Solution to the temperature gradient in a rod at time $t = 0.05$ using the numerical method of finite difference. For both Δx, the corresponding Δt was set to $\Delta t = \Delta x^2/2$. The gray line marked "exact" shows the analytical solution.

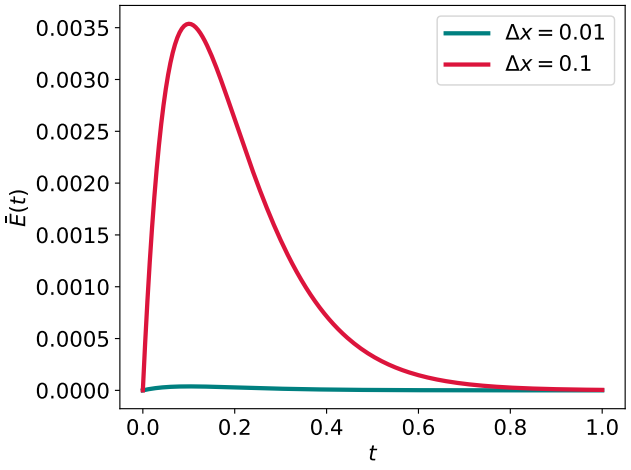

FIG. 4: The mean absolute error, as given in eq. (15), between the analytical solution and the solution computed by the finite difference method. For both Δx, the corresponding Δt was set to $\Delta t = \Delta x^2/2$.

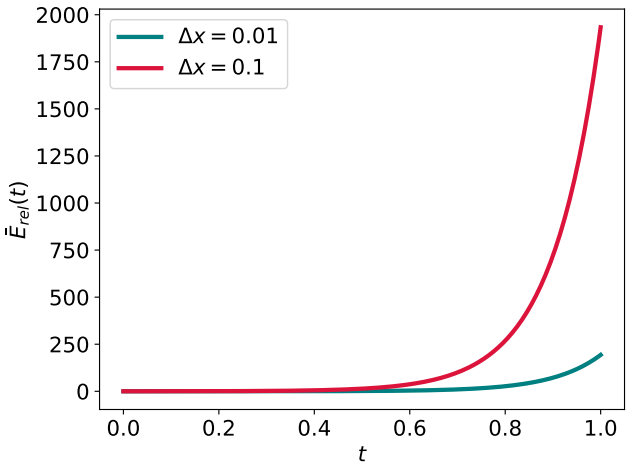

FIG. 5: The mean relative error, as given in eq. (16), between the analytical solution and the solution computed by the finite difference method. For both Δx, the corresponding Δt was set to $\Delta t = \Delta x^2/2$.

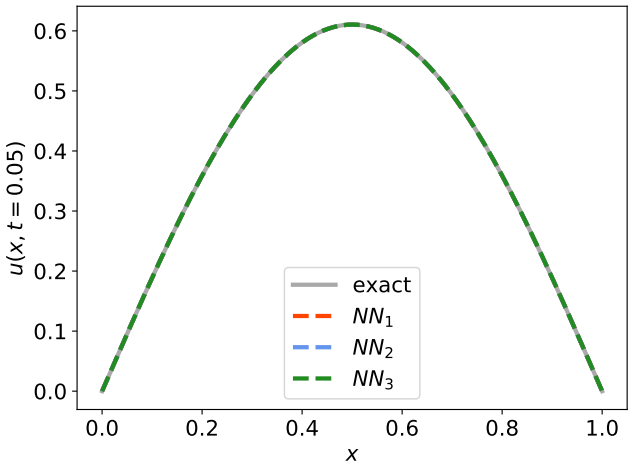

FIG. 6: Solution to the temperature gradient in a rod at time $t = 0.05$ using three identical artificial neural networks. The networks contained two hidden layers containing 20 neurons each and used the Sigmoid function as activation function. The activation function of the output layer was linear. The gray line marked "exact" shows the analytical solution.

The mean squared error of the entire simulation was calculated. The mean squared error was found to be $3.2 \cdot 10^{-6}$ when $\Delta x = 0.1$ and $3.4 \cdot 10^{-10}$ when $\Delta x = 0.01$. In both cases Δt was set to $\Delta t = \Delta x^2/2$.

2. *Heat Flow Equation by Artificial Neural Network*

In the study of performance of the artificial neural network solving the heat flow equation of the temperature gradient in a rod, all parameters were held constant, after finding a network with relatively good results, through trial and error. The neural networks used the same learning rate in the ADAM optimizer ($\eta = 0.01$) and ran the same amount of back propagation iterations ($2 \cdot 10^4$).

The figures below (6 - 7) show the calculated solution to the temperature gradient in a rod at the two chosen points in time ($t = 0.05$ and $t = 0.75$), using artificial neural networks.

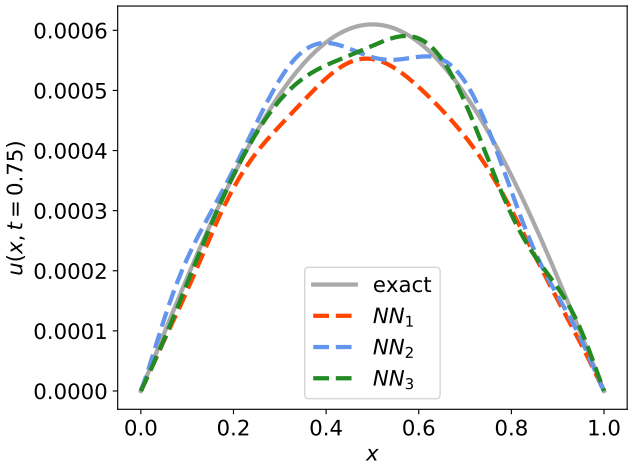

FIG. 7: Solution to the temperature gradient in a rod at time $t = 0.75$ using three identical artificial neural networks. The networks contained two hidden layers containing 20 neurons each and used the Sigmoid function as activation function. The activation function of the output layer was linear. The gray line marked "exact" shows the analytical solution.

A study of the error of the solutions of the artificial neural network compared to the analytical solution was made. The error is shown as a function of time on the figures below (8 - 9).

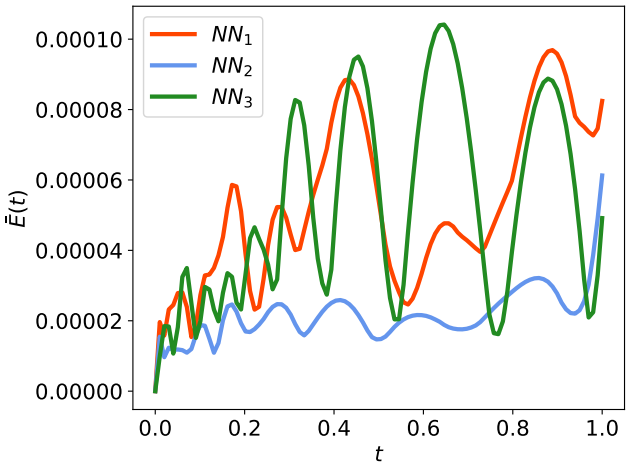

FIG. 8: The mean absolute error, as given in eq. (15), between the analytical solution and the solution computed by the artificial neural network. The networks contained two hidden layers containing 20 neurons each and used the Sigmoid function as activation function.

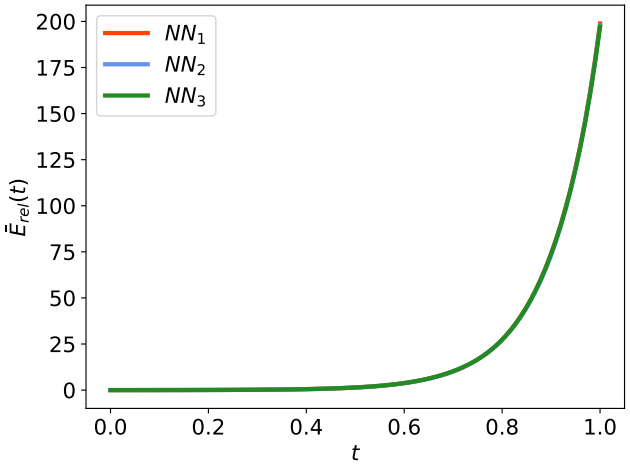

FIG. 9: The mean relative error, as given in eq. (16), between the analytical solution and the solution computed by the artificial neural network. The networks contained two hidden layers containing 20 neurons each and used the Sigmoid function as activation function.

In table (I), shown below, the MSE-scores of the three artificial neural networks solving the heat flow equation of the temperate gradient in a rod are shown.

TABLE I: Comparing MSE-scores of the three artificial neural networks. The networks contained two hidden layers containing 20 neurons each and used the Sigmoid function as activation function. The column Cost shows the value of the cost function as shown in eq. (24). The two next columns show the MSE scores calculated using eq. (17), Train indicating that the training data was used and Test indicating that test data was used to calculate the score.

Cost	Train	Test
$2.2 \cdot 10^{-5}$	$4.8 \cdot 10^{-9}$	$4.8 \cdot 10^{-9}$
$6.4 \cdot 10^{-5}$	$8.1 \cdot 10^{-10}$	$8.4 \cdot 10^{-10}$
$3.7 \cdot 10^{-5}$	$4.8 \cdot 10^{-9}$	$4.7 \cdot 10^{-9}$

The mean training time of the networks was 8.9 minutes.

B. Finding Eigenvalues

The neural network used to solved the heat flow equation was used to solve an eigenvalue problem for a 6×6 symmetric matrix, using (26) as cost function. As an example, 30 runs was done on the following matrix:

$$\begin{pmatrix} 0.1915 & -0.2514 & -0.6939 & 0.1231 & -0.1355 & 0.3932 \\ -0.2514 & -0.5086 & -0.0317 & -0.0454 & 0.3932 & 0.4711 \\ -0.6939 & -0.0317 & 0.5567 & -0.8125 & -0.2213 & 0.7452 \\ 0.1231 & -0.0454 & -0.8125 & 0.1440 & 0.3555 & -0.7431 \\ -0.1355 & 0.3932 & -0.2213 & 0.3555 & -0.5750 & -0.5177 \\ 0.3932 & 0.4711 & 0.7452 & -0.7431 & -0.5177 & -0.1931 \end{pmatrix}.$$

The eigenvalues for this matrix was $\{-1.5277, -0.7867, -0.5991, -0.1104, 0.6744, 1.9652\}$, found using `NumPy`'s `eigh` method for diagonalization. Figure (10) shows the distribution of the number of each eigenvalue found from the 30 runs. The runtime for the 30 runs was 364 seconds.

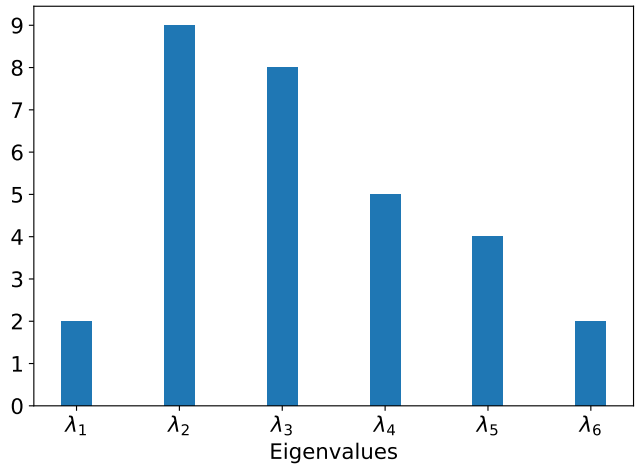

FIG. 10: Number of each eigenvalue found after 30 runs of the neural network, with $\eta = 0.1$, number of epochs = 10, number of iterations per epoch = 4000. The eigenvalues are ordered as $\lambda_1 < \lambda_2 < \lambda_3 < \lambda_4 < \lambda_5 < \lambda_6$.

V. DISCUSSION

A. The Heat Flow Equation

The heat flow equation was approximated in different ways in this project. We will start by discussing the performance of the finite difference method. Then look at how well the artificial neural network solves the partial differential equation. Lastly, we will compare the two methods.

1. *Solution by Finite Difference*

The performance of the finite difference scheme heavily depends on the step lengths Δx and Δt. In this project, Δt was set to the limit of the stability criterion, which is $\Delta t \leq \Delta x^2/2$. In figures (2) and (3) the finite difference scheme using $\Delta x = 0.01$ seems to overlap the analytical solution perfectly. The simulations using $\Delta x = 0.1$ shows an increasing in error from $t = 0.05$ to $t = 0.75$.

The figures showing the error as a function of time should be observed together. Figure (4) show the mean absolute error, and from this figure it seems like the largest error is before $t = 0.2$. Note that the absolute error of $\Delta x = 0.01$ has the exact same curve as $\Delta x = 0.1$, only that the peak is no higher than $4 \cdot 10^{-5}$. The next figure (5) shows the relative error, and from this figure we can see that the error is not actually decreasing, but increasing with time. The exponential growth in relative error is much due to the system approaching the equilibrium state, where all values approach zero. The two pictures combined show the growth in error from initial condition.

The smaller step length $\Delta x = 0.01$ preformed better in every way. This was also seen in the mean squared error of the entire simulation, where $\Delta x = 0.1$ had a score of $3 \cdot 10^{-6}$, while $\Delta x = 0.01$ had a score of $3 \cdot 10^{-10}$. This result was expected.

2. *Solution by Neural Networks*

Figure (6) shows the system at time $t = 0.05$. At this point, no error from the analytical solution is visible for any of the artificial neural networks used. In figure (7), showing the rod at $t = 0.75$, we can see a significant difference from the exact solution.

The error seem to fluctuate for all three neural networks, as can be seen from figure (9). Here we also see that the second neural network, labeled NN_2, seems to outperform the two other networks.

This is also visible in table (I), in both the scores from the cost function showing how well the heat flow equation was fulfilled and in the mean squared error scores comparing to the analytical function using both training data and test data. The cost function after training yielded scores around 10^{-5}. The mean squared error score was worse for the first and third neural network ($\sim 10^{-9}$) than for the second ($\sim 10^{-10}$). An interesting thing to note is that the neural networks scored equally well on the test data compared to their training data.

In the relative error, shown in figure (5), the three neural networks seem indistinguishable. All three grow exponentially as the system approaches the equilibrium state.

An important thing to conclude from these differences is the stochastic element of these computations are significant. It can be observed that the convergence of the cost function is clearly impacted by the randomly initialized weights between neurons and biases of the neurons.

A more rigorous study for optimizing the architecture of the neural network and longer periods of learning could without a doubt yield better results than the ones presented in this project.

3. *Comparing Solutions*

From comparing the solutions at $t = 0.75$ from numerical calculations in figure (3) and that of the neural networks in figure (7) we see that the finite difference method using $\Delta x = 0.01$ is the only approximation that has no visible error to the exact solution.

In the case of a temperature gradient in a rod, the neural network did not outperform the classical numerical implementation. The results from finite difference using $\Delta x = 0.01$ gave the most predictable and accurate results. The error seen in the figures (4), (5), (8) and (9) consistently show that the finite difference method using $\Delta x = 0.01$ was the approximation closest to the analytic solution, followed by the artificial neural network.

The runtime of the neural network was vastly greater than that of the finite difference. While finite difference gave instant results (no noticeable runtime), the neural network needed almost 10 minutes to learn with the given precision.

The strength of the solution using artificial neural networks is that there is no need for a stability criterion in the step length. Given a partial differential equation where there is need for a small step length to ensure stability in the numerical solution, but the problem needs to propagate over longer *distances* in e.g time, the numerical error will accumulate over time, as each time step depends on previous calculations. In this case, the numerical method will also use considerable time to compute a solution. This problem of stability is avoided when using the artificial neural network, which looks at each step individually. For a partial differential equation where the numerical solution either has a very strict stability criterion or is unconditionally unstable, the neural network may be a superior solution to the problem.

B. Finding Eigenvalues

Using our implementation of neural network ODE solver, we managed to find all eigenvalues of randomly generated symmetric 6×6 matrices. As seen in figure (10) the eigenvalue the network converges to is random, due to the random initialization of the weights of the neural network. However, running the same matrix multiple times through the network eventually yielded all the eigenvalues. This is in contrast to the method proposed by Yi *et al.* [4] which only found the smallest and largest eigenvalues.

While extending the neural network ODE solver to handle the eigenvalue problem was trivial, it is difficult to argue in favor of using this method on real symmetric matrices instead of `NumPy`'s `eigh` method, which is optimized to compute eigenvalues of real symmetric matrices. The runtime of the 30 runs in our example was 364 seconds, while the `eigh` runtime was less than 1 second. In principle one could be fortunate and only need 6 runs of the neural network to get all 6 eigenvalues, but the runtime would still be far outmatched by `eigh`.

Some modifications to the network to disfavor finding identical eigenvalues in consecutive runs is probably possible, but was not implemented in this project.

VI. CONCLUSION

As expected, using smaller step lengths in the finite difference scheme yielded better results.

We conclude that the artificial neural networks are able to predict the solution to a partial differential equation, as error was quite low (max absolute error 0.0001 and mean squared error $\sim 10^{-9}$) with possibilities for better results if letting the network learn over longer time and/or optimizing the network parameters.

The artificial neural network was considerably more complicated and time consuming to implement, requiring several iterations to optimize variables such as number of layers, numbers of neurons, activation functions, learning rate and more. The neural network also used far longer to train (nearly 10 minutes) compared to how long the finite difference method needed to compute the results. The artificial neural network was also outperformed by the finite difference method using $\Delta x = 0.01$ in terms of accuracy in the case of the temperature gradient in a rod.

In the case of the heat flow equation, the finite difference method is a superior method to the artificial neural network. The artificial neural network *can* solve partial differential equations, and can therefore be a viable option for differential equations where the numerical implementation is either unconditionally unstable or will accumulate significant numerical error due to strict stability criterion over long propagation.

An artificial neural network can also be used to find all eigenvalues for symmetric matrices, but we can present no argument in favor of using this method over standard diagonalization methods readily available from known linear algebra packages. Both implementation and computation is heavily time consuming compared to the rather straight forward usage of other methods.

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, 2015.

[3] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[4] Zhang Yi, Yan Fu, and Hua Jin Tang. Neural networks based approach for computing eigenvectors and eigenvalues of symmetric matrix. *Computers & Mathematics with Applications*, 47(8):1155 – 1164, 2004.