Project 3: Solving partial differential equations with neural networks

Course: FYS-STK4155

Semester: Autumn 2020

Name: Sander Losnedahl

Abstract

This project first considers three different ways of solving the one dimensional heat equation. An analytical solution is first proposed followed by a numerical implementation in the form of the forward Euler scheme. It is shown that the forward Euler scheme can become arbitrarily close to the analytical solution for all time and space when the step sizes in time and space is decreased.
Then, a neural network algorithm is introduced in order to find a solution to the one dimensional heat equation. It is shown that the algorithm can approximate the analytical solution when the number of feed forward/back propagation iterations become sufficiently many. The approximation is great for earlier times, but the approximation starts to deviate at later times (lower temperatures), as the cost function was not sufficiently reduced for such small values.
Lastly, this project utilizes the above mentioned neural network code in order to find the smallest/largest eigenvalue of a real, symmetric and square matrix. The cost function used was proposed by Yi et.al. in the paper *Neural Networks Based Approach Computing Eigenvectors and Eigenvalues of Symmetric Matrix* and the results from the paper was shown to be reproducible (Yi et.al., 2004). The results obtained from this method is then shown to be comparable with the one obtained from a generic linear algebra solver (Python package numpy in this case).

Introduction

This project first solves the one dimensional heat equation (function of time and one direction in space) both analytically and numerically. The numerical part consists of both a forward Euler scheme and a neural network algorithm. The two numerical solutions are then compared to the analytical solution. Then, the same neural network algorithm is slightly adjusted in order for the algorithm to find the smallest/largest eigenvalue of a $6 * 6$ real, symmetric and randomly generated matrix.
We start by taking a look at the heat equation and finding an analytical solution to this partial differential equation. A forward Euler scheme is proposed and implemented together with the analytical solution and the results of the two implementations are compared. The comparison is then repeated for different step sizes of the forward Euler scheme.
Then, a dense neural network algorithm is implemented in order to solve the heat equation using the Adam optimizer with Tensorflow. The algorithm itself is first discussed and the cost function is defined. The solution obtained from the neural network solution is then compared with the analytical solution.
The report then considers a 6*6 real, symmetric and randomly generated matrix where the goal is to find the smallest/largest eigenvalue. The neural network algorithm previously mentioned is reused with slight adjustments and the resulting eigenvalue is compared with an eigenvalue obtained from a generic eigenvalue solver (numpy was used in this case).
The results found in this project is then discussed further together with other literature and a conclusion is drawn with possible improvements to the methods utilized in this project.

Preliminaries

The Github page can be found at `https://github.com/sanderwl/FYS-STK4155/tree/master/Project3` and includes all code and figures used in this report.
Tensorflow is a central Python package that is needed for the much of the neural network code in this project. I suggest you install Tensorflow with **Anaconda** as this proved much easier than with some other local interpreter.

Exercise 1a): An analytical approach to the one dimensional heat equation

Defining the heat equation

In this exercise we want to mathematically describe how heat can transfer within a rod of length L over a given time interval t. The rod will initially be heated at time $t = 0$ with some distribution, but after this point in time, the rod will cool off and the heat will decay as a function of time. How the heat changes as function of time and space is given by the heat equation as seen in equation 1.

$$\frac{\partial u(x,t)}{\partial t} = \frac{K_0}{c\rho}\frac{\partial^2 u(x,t)}{\partial x^2} \tag{1}$$

where $u(x,t)$ is the temperature at a specific time t and position x. The term $\frac{K_0}{c\rho}$ is the thermal diffusivity which consists of thermal conductivity K_0 divided by the specific heat capacity c times the density of the material ρ (Venkanna, 2010). This quantity is a measure of the rate of heat transfer from the heated part of the rod to the cold part of the rod.
Equation 1 is a partial differential equation (PDE) which means that we need an initial condition and two boundary conditions in order to solve the equation. The initial condition can be interpreted as how much the rod is heated at $t = 0$ while the boundary conditions can be interpreted as how the heat reacts to the ends of the rod at $x = 0$ and $x = L$. The boundary conditions in this case is given by equations 2 and 3.

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

$$u(L,t) = 0 \tag{3}$$

We can set the initial condition such that the rod is heated the most at the middle and least at the ends. Such a condition can be described by a sine function as given in equation 4.

$$u(x,0) = sin(\pi x) \tag{4}$$

One can observe from equation 4 that a rod of length 1 is heated the most at $L/2 = 0.5$ and is not heated at all at the ends of the rod.

Analytical solution to the heat equation

We can now split equation 1 into its spatial and temporal components such that

$$u(x,t) = X(x)T(t) \tag{5}$$

Equation 5 can then be solved for x and t respectively. We start by taking the partial derivatives of equation 5 with respect to both t and x.

$$\begin{aligned} u_t &= X(x)T'(t) \\ u_{xx} &= X''(x)T(t) \end{aligned} \tag{6}$$

Equation 1 then takes the form

$$\begin{aligned} u_t &= u_{xx} \\ X(x)T'(t) &= X''(x)T(t) \\ \frac{X''(x)}{X(x)} &= \frac{T'(t)}{T(t)} = \lambda \end{aligned} \tag{7}$$

where λ is some constant. We can first solve equation 7 for its spatial part X such that

$$\begin{aligned} \frac{X''(x)}{X(x)} &= \lambda \\ X''(x) - \lambda X(x) &= 0 \end{aligned} \tag{8}$$

Equation 8 is recognized as a Sturm-Liouville problem when λ represents the eigenvalues of equation 8 (Hancock, 2006). This means that the solutions to equation 8 is given by equation 9.

$$X_n(x) = b_n sin(n\pi x) \tag{9}$$

where n is an eigenfunction given by $n = \sqrt{\lambda/\pi}$ where λ are the eigenvalues of the Sturm-Liouville problem. Solutions then arise at $\lambda = \pi^2 n^2$ and from equation 7 the solution with respect to $T'(t)$ is found.

$$\begin{aligned} \frac{T'(t)}{T(t)} &= \lambda \\ T'(t) &= \lambda T(t) \end{aligned} \tag{10}$$

which has the solution (Hancock, 2006)

$$T'(t) = e^{\pi^2 n^2 t} \tag{11}$$

We can then find the solution for $u(x,t)$ by combining equation 9 and equation 11 such that

$$u_n(x,t) = b_n sin(n\pi x) e^{\pi^2 n^2 t} \tag{12}$$

The subscript n denotes the individual eigenfunctions that solve the heat equation, however, most of them do not satisfy the initial condition in equation 4. The sum of these functions may satisfy the initial condition and thus, equation 12 becomes

$$u_n(x,t) = \sum_{n=1}^{\infty} b_n sin(n\pi x) e^{\pi^2 n^2 t} \tag{13}$$

One can observe from equation 13 that the exponential term decreases as n increases. Therefore, a reasonable approximation may be to only utilize the first term of the sum such that equation 13 can be written as

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

where the constant b_n has been set to one. This is the final solution of the heat equation that will be implemented in the code.

Exercise 1b: A numerical approach to the heat equation

The forward Euler scheme

The heat equation is a partial differential equation that can be solved numerically using the forward Euler scheme (sometimes referred to as the explicit Euler method). This algorithm aims to incrementally approximate the gradient of spatial and temporal parts of the heat equation by utilizing the definition of the derivative as seen in equations 15 and 16.

$$u_t \approx \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} \tag{15}$$

$$u_{xx} \approx \frac{u(x + \Delta x, t) - 2u(x, t) + u(x - \Delta x, t)}{\Delta x^2} \tag{16}$$

where Δx and Δt represents the incremental step size in space or time respectively. Since the initial and boundary conditions are known, the forward Euler algorithm can incrementally approximate the change in heat from the middle of the rod and out towards the boundary. Equations 15 and 16 can be combined using the heat equation from equation 1 such that

$$\begin{aligned} u_t &= u_{xx} \\ \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} &= \frac{u(x + \Delta x, t) - 2u(x, t) + u(x - \Delta x, t)}{\Delta x^2} \\ \frac{u_j^{n+1} - u_j^n}{\Delta t} &= \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^2}{\Delta x^2} \end{aligned} \tag{17}$$

where the subscript j denotes the step in space and the subscript n denotes the step in time. We can solve equation 17 for u_j^{n+1} such that we can calculate the heat incrementally forward in time. Thus, equation 17 becomes

$$\begin{aligned} u_j^{n+1} &= \Delta t(\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^2}{\Delta x^2}) + u_j^n \\ u_j^{n+1} &= (1 - 2\frac{\Delta t}{\Delta x^2})u_j^n + \frac{\Delta t}{\Delta x^2}(u_{j+1}^n + u_{j-1}^n) \end{aligned} \tag{18}$$

Equation 18 is only stable for $\frac{\Delta t}{\Delta x^2} \leq 0.5$, so the implementation needs to adjust Δt according to Δx. The implementation of equation 18 allows us to solve the heat equation incrementally and the results are then compared to the analytical solution in the following section.

Comparing the analytical and numerical solutions of the heat equation

We now compare how the heat translates throughout the rod at a given time for both the analytical implementation and for the forward Euler implementation. It was found that the heat decayed quickly as a result of the constant b_n being set to one in equation 14. Therefore, good time intervals of observing the heat was found to be at $t = 0$ (initial heat), $t = 0.5$ and $t = 1$. Figures 1, 2 and 3 shows the heat distribution in the rod at these given times for $\Delta t = 0.005$ and $\Delta x = 0.1$ for both the analytical and forward Euler implementations (and their difference).

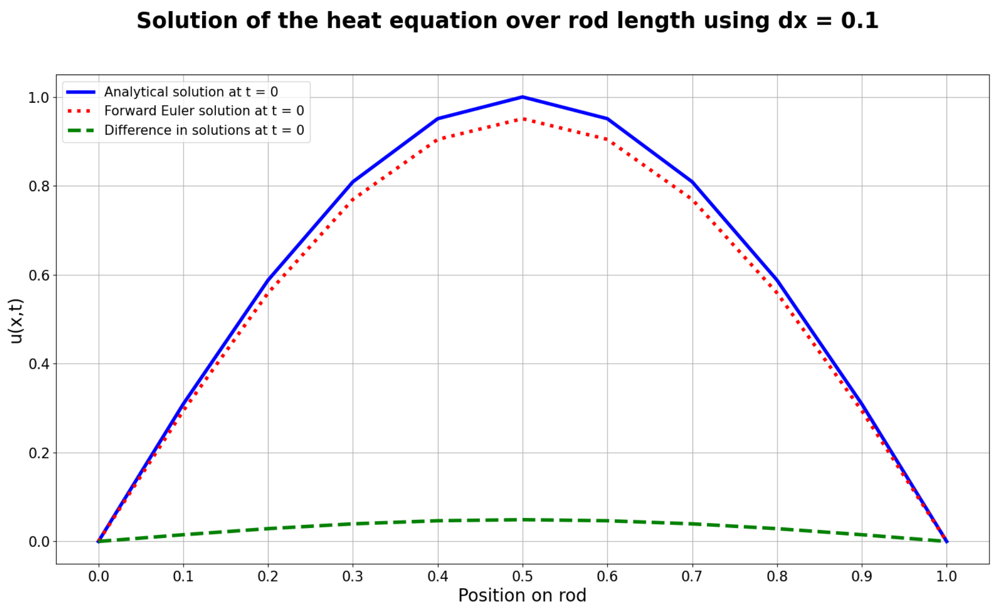

Figure 1: Heat distribution within the rod for both analytical (blue) and forward Euler (red) implementations. The heat is measured at $t = 0$ using step sizes of $\Delta t = 0.005$ and $\Delta x = 0.1$.

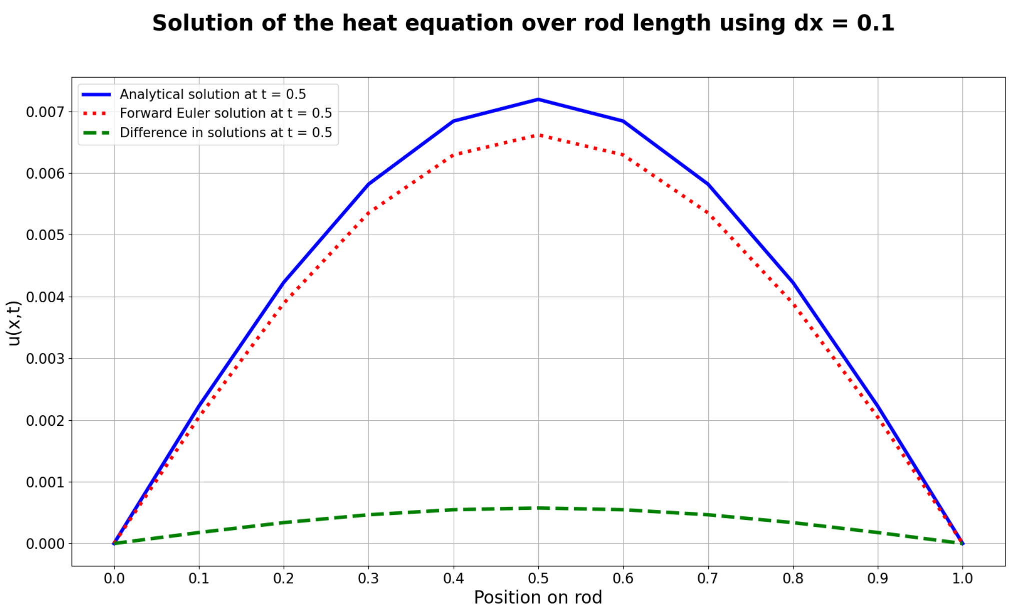

Figure 2: Heat distribution within the rod for both analytical (blue) and forward Euler (red) implementations. The heat is measured at $t = 0.5$ using step sizes of $\Delta t = 0.005$ and $\Delta x = 0.1$.

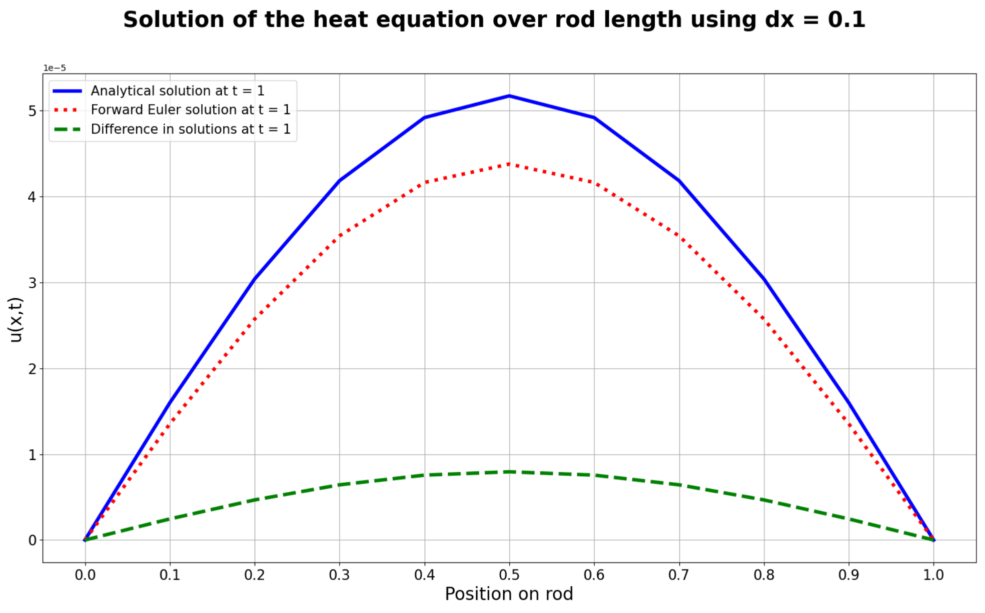

Figure 3: Heat distribution within the rod for both analytical (blue) and forward Euler (red) implementations. The heat is measured at $t = 1$ using step sizes of $\Delta t = 0.005$ and $\Delta x = 0.1$.

One first observes that the rod is heated the most in the middle at $x = 0.5$, which is according to the initial condition of equation 4. One can also observe that the forward Euler lies pretty close to the analytical solution. The difference between the analytical solution and the forward Euler solution decreases as time increases (the y-axis becomes smaller, so don't get fooled). Additionally, one can observe that the heat decays fairly

quickly as most of the heat is gone within 1 second.
We now decrease the step size in space from $\Delta x = 0.1$ to $\Delta x = 0.01$, and this means that we need to adjust the step size in time according to $\frac{\Delta t}{\Delta x^2} \leq 0.5$. Therefore, the new step size in time becomes $\Delta t = 0.5 * 0.01^2 = 0.00005$. Figures 4, 5 and 6 shows the same as the above figures, but using the new step sizes.

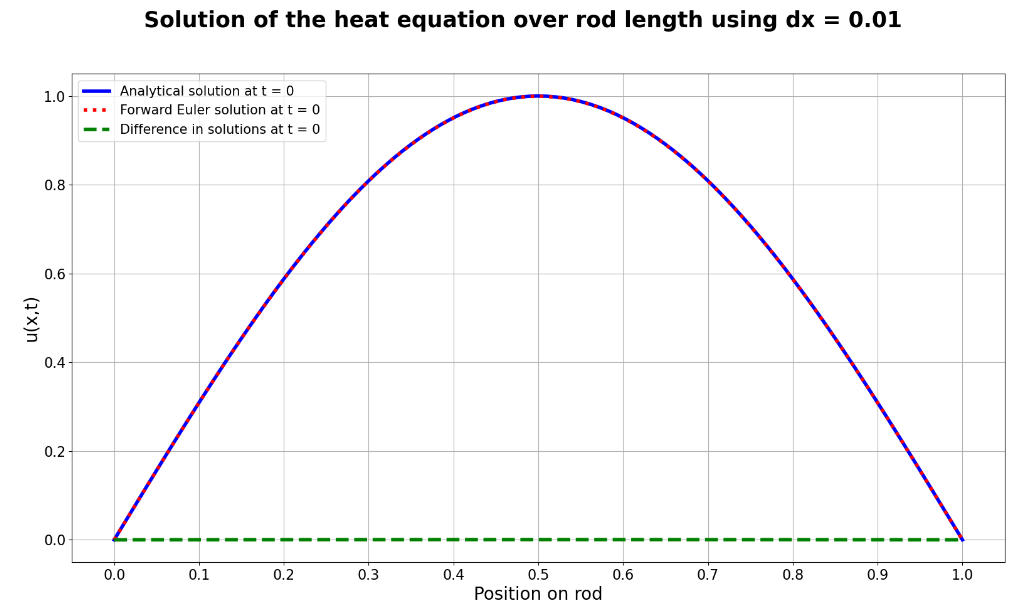

Figure 4: Heat distribution within the rod for both analytical (blue) and forward Euler (red) implementations. The heat is measured at $t = 0$ using step sizes of $\Delta t = 0.00005$ and $\Delta x = 0.01$.

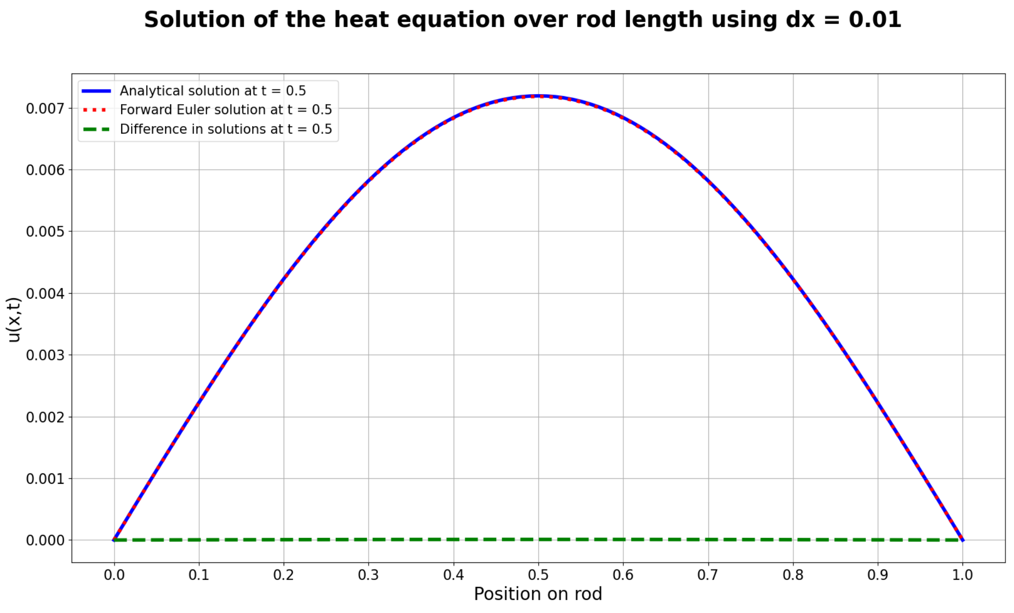

Figure 5: Heat distribution within the rod for both analytical (blue) and forward Euler (red) implementations. The heat is measured at $t = 0.5$ using step sizes of $\Delta t = 0.00005$ and $\Delta x = 0.01$.

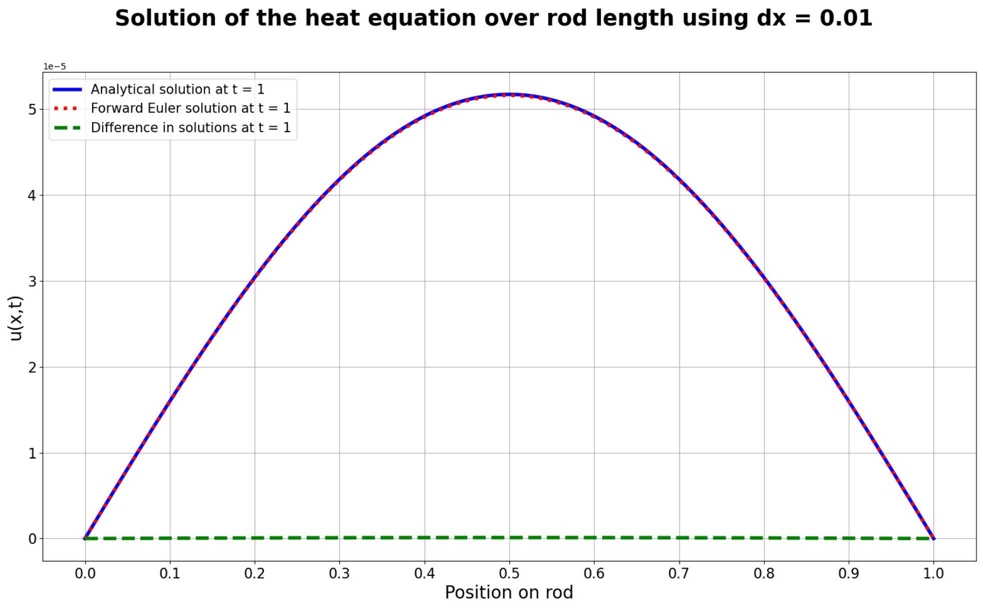

Figure 6: Heat distribution within the rod for both analytical (blue) and forward Euler (red) implementations. The heat is measured at $t = 1$ using step sizes of $\Delta t = 0.00005$ and $\Delta x = 0.01$.

It is now observed that the forward Euler implementation matches the analytical implementation perfectly at all times. The reason for this is because the forward Euler is based on the definition of the derivative which becomes a better approximation of the true derivative when the step sizes are small. This means further that the forward Euler implementation can accurately represent the heat distribution within a rod at all times and places.

Exercise c): Solving the heat equation using neural networks

Neural network implementation

The concepts and details of a neural network was discussed in Project 2 which can be found at `https://github.com/sanderwl/FYS-STK4155/tree/master/Project2/Project2`. Many of the concepts are the same, that is, we pass an input through a network of hidden layers and neurons and an output is produced. This process is then repeated where the goal is to minimize some cost function using gradient descent and weight updates. However, the implementation in this project is slightly different.
The neural network still aims to minimize some cost function which is this case is the MSE given by equation 19.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\text{prediction error})^2 \tag{19}$$

Since $\frac{\partial u(x,t)}{\partial t} - \frac{\partial^2 u(x,t)}{\partial x^2} = 0$ solves the heat equation, it can be utilized as the prediction error and equation 19 then becomes

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\frac{\partial u(x,t)}{\partial t} - \frac{\partial^2 u(x,t)}{\partial x^2})^2 \tag{20}$$

The goal of the neural network is to tune the weights such that the output matches the set of boundary conditions given in equations 2 and 3. For this, a trial solution is proposed. This solution is not guaranteed to minimize the cost function in equation 20, in fact, it is rare that such a solution minimize the cost function. Therefore, weights are tuned through gradient descent and different trial solutions are then proposed according to the current weights. For each update, the cost function gets closer to its minimal value and a more accurate solution to the heat equation is found. The trial solution g used in this project is given by equation 21.

$$g = (1-t) * sin(\pi x) + x(1-x)t * \text{output} \tag{21}$$

where $sin(\pi x)$ is the initial condition and output is the output from the neural network in which the gradient descent must handle.
The neural network is then trained by sending the two variables x and t through the network followed by a weight update through gradient descent on the cost function. This process happens many times and the final solution as function of the weights should reflect the analytical solution.

Tensorflow implementation

A neural network algorithm was implemented from scratch using what I have learned in this course. However, this algorithm proved way too slow and was thus deemed unfit for this exercise as time is limited. Instead, Tensorflow algorithms were implemented as these yielded fast, stable and good results. Additionally, Tensorflow is easy to handle and has a lot of nice features to play around with. In particular, the Tensorflow version of the Adam optimization algorithm proved to yield great results. The Adam optimization algorithm is really a combinations of stochastic gradient descent with AdaGrad and RMSprop (Kingma and Ba, 2014). The algorithm is essentially stochastic gradient descent that utilizes both first and second moment adaptive learning rates (RMSProp) and scaling to the learning rate for each feed forward/back propagation (AdaGrad). The Adam has shown to be a robust and fast approach to deep learning, with some flaws, in the paper by Kingma and Ba (Kinga and Ba, 2014). The algorithm including the Adam optimization algorithm is described below.

```
Algorithm 1: Neural network algorithm
  Result: A trained model equivalent to solving the heat equation
  Declare x and t in the Tensorflow format;
  Define the cost function according to equations 20;
  Define the initial trial solution according to equation 21;
  Set up the dense layered neural network with the Adam optimizer;
  for iterations do
      Feed forward of x and t;
      Gradient descent on the cost function with Adam optimizer;
      Back propagation to update weights;
  end
  Compare neural network result with the analytical solution
```

In order to train the neural network described above, some input parameters are needed. Through experimentation it was found that many different configurations (set of input parameters) yielded great results. In the end however, the parameters of table 1 were chosen.

Table 1: Parameters of the neural network.

Δ t	0.1
Δ x	0.1
Number of layers	3
Number of neurons in each layer	10
Number of iterations	1000000
Initial learning rate	0.001
Activation function	Sigmoid

Since we are doing a stochastic gradient descent algorithm (Adam optimizer) we have to set some initial learning rate. The number of iterations denotes how many times we do the feed forward/back propagation loop. The other parameters were found through trial and error. These parameters could have been optimized like in Project 2, but the results have shown to be good without exact optimization.

Comparing the analytical and neural network solutions of the heat equation

After training the neural network as described in algorithm 1 with the parameters from table 1, we can pass x and t through the trained model such that the output is equivalent with solving the heat equation. When this is done, the two-dimensional heat distribution (time and one spatial dimension) can be plotted as done in figures 7 and 8.

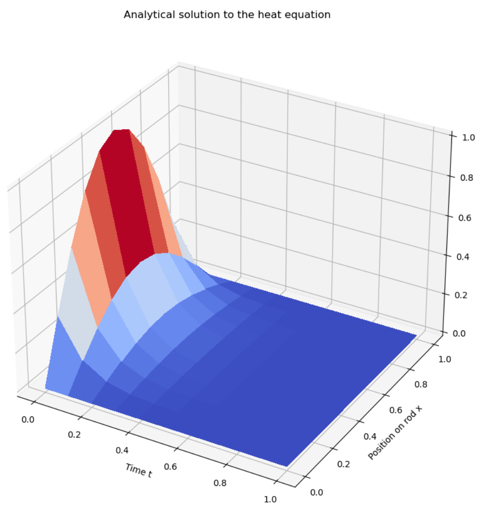

Figure 7: Heat distribution as function of space and time for the analytical implementation.

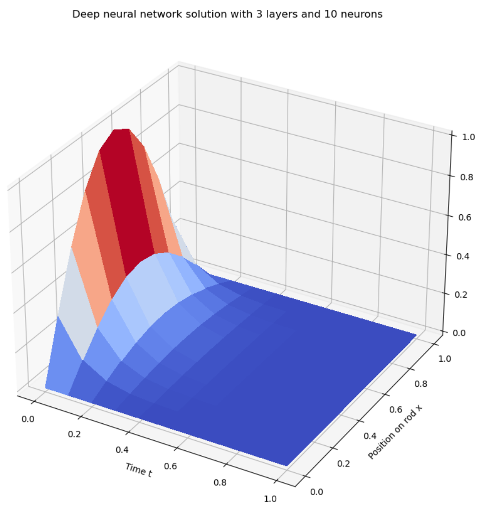

Figure 8: Heat distribution as function of space and time for the neural network implementation.

One can observe that the two implementations in figures 7 and 8 are pretty similar. In order to get a better comparison, we can plot the heat distribution at constant times $t = 0$ (initial heat), $t = 0.5$ and $t = 1$ just like before.

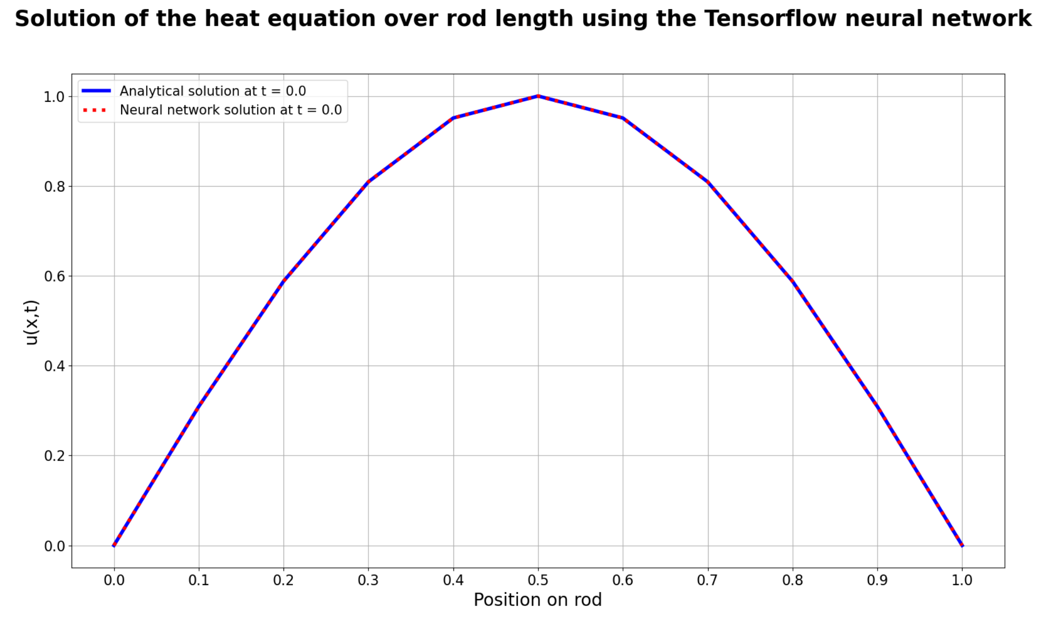

Figure 9: Heat distribution within the rod for both analytical (blue) and neural network (red) implementations. The heat is measured at $t = 0$ using the parameters of table 1.

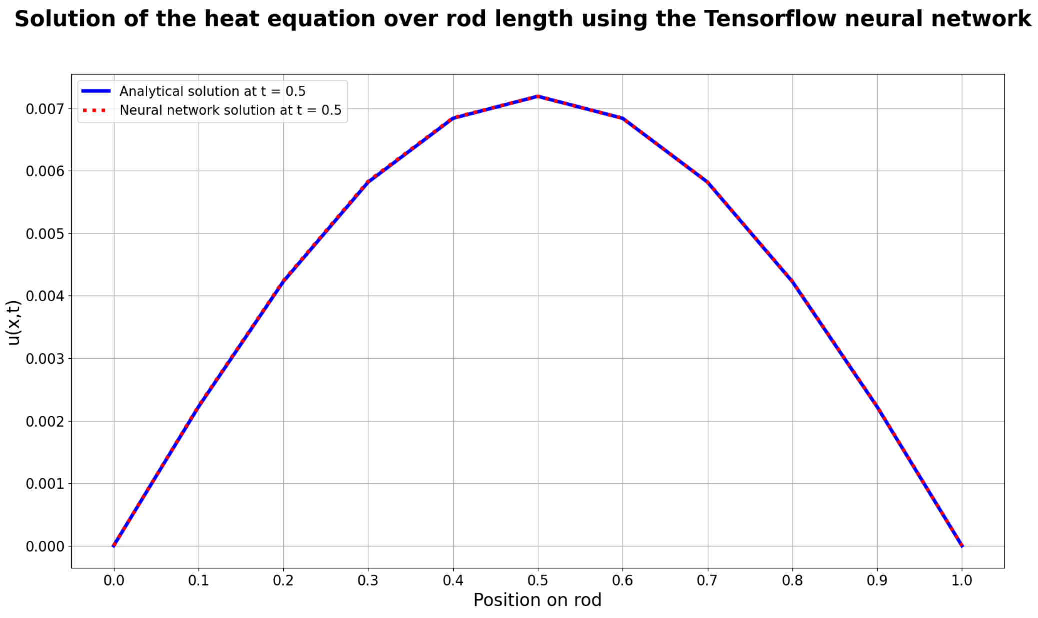

Figure 10: Heat distribution within the rod for both analytical (blue) and neural network (red) implementations. The heat is measured at $t = 0.5$ using the parameters of table 1.

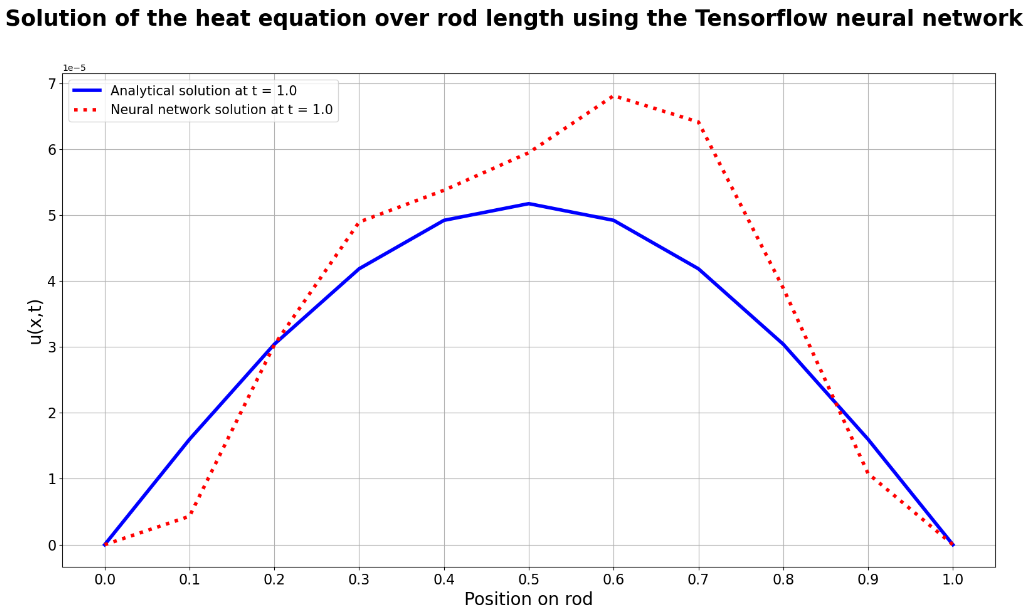

Figure 11: Heat distribution within the rod for both analytical (blue) and neural network (red) implementations. The heat is measured at $t = 1$ using the parameters of table 1.

One can observe that the initial heat distribution in figure 9 is equal for the analytical and neural network implementations. After 0.5 seconds, the two implementations are still equivalent. At $t = 1$ however, the two implementations are essentially zero, but the neural network has now deviated from the analytical solution. This was a recurring problem with the neural network implementation regardless of input parameters. When the temperature started to approach zero, the neural network increasingly deviated from the analytical solution. However, since the deviation is very small, the result is deemed acceptable. Additionally, one can observe that the temperature at the different times are equivalent to the ones found by the forward Euler scheme.

Exercise d): Finding the smallest/largest eigenvalue using neural networks

In this exercise we aim to find the smallest/largest eigenvalue with a method proposed by Yi et.al. in the paper *Neural Networks Based Approach Computing Eigenvectors and Eigenvalues of Symmetric Matrix* (Yi et.al., 2004). The method utilizes a neural network set up similar to what is implemented in the previous exercise, but with a cost function given by equation 22.

$$\frac{dx(t)}{dt} = -x(t) + f(x(t)) \tag{22}$$

where $x(t)$ is the state of the network (where and when the input is propagated through the network) and $f(x(t))$ is given by

$$f(x(t)) = [x^T xA + (1 - x^T Ax)I]x \tag{23}$$

Here, A is a real, symmetric and square matrix, I is the identity matrix and T marks the transpose. A is in this case defined by $k(Q^T + Q)/2$ which ensures symmetry, where Q is a real, randomly generated square matrix. k is in this case either -1 or 1 where $k = -1$ yields the smallest eigenvalue of the matrix and $k = 1$ yields the largest eigenvalue of the matrix.
Most of the neural network code is reused from the previous exercise as described by algorithm 1, but the cost function is changes to equation 22. Additionally, we utilize the input parameters of table 1. The algorithms stops iterating if the cost function is less that some threshold set to 0.00001 in this case. The threshold is very low because the neural network results showed to be very sensitive to errors larger than this threshold.
The matrix A was randomly generated and was in this run set to

$$A = \begin{bmatrix} 0.07630829 & 0.64051963 & 0.40967518 & 0.8273356 & 0.94355894 & 0.37167248 \\ 0.64051963 & 0.07205113 & 0.16718766 & 0.26239086 & 0.40619972 & 0.64725246 \\ 0.40967518 & 0.16718766 & 0.2881456 & 0.75507122 & 0.36839897 & 0.41225433 \\ 0.8273356 & 0.26239086 & 0.75507122 & 0.9501295 & 0.49035637 & 0.51294554 \\ 0.94355894 & 0.40619972 & 0.36839897 & 0.49035637 & 0.66901324 & 0.41682162 \\ 0.37167248 & 0.64725246 & 0.41225433 & 0.51294554 & 0.41682162 & 0.83791799 \end{bmatrix}$$

which is a real and symmetric, square matrix. Running the neural network with $k = 1$ then yields an eigenvector ν_{max} corresponding to the largest eigenvalue λ_{max} of

$$\nu_{max} = [-0.51370594, -0.35445244, -0.39575076, -0.60860367, -0.51998609, -0.48656764] \tag{24}$$

and a normalized eigenvector of

$$\nu_{max,norm} = [-0.43053478, -0.29706509, -0.33167704, -0.51006817, -0.43579814, -0.40779028] \tag{25}$$

with a corresponding eigenvalue of

$$\lambda_{max} = 3.1220778592498455 \tag{26}$$

The neural network gets closer and closer to the correct eigenvector/eigenvalue for each iteration and figure 12 shows exactly this convergence.

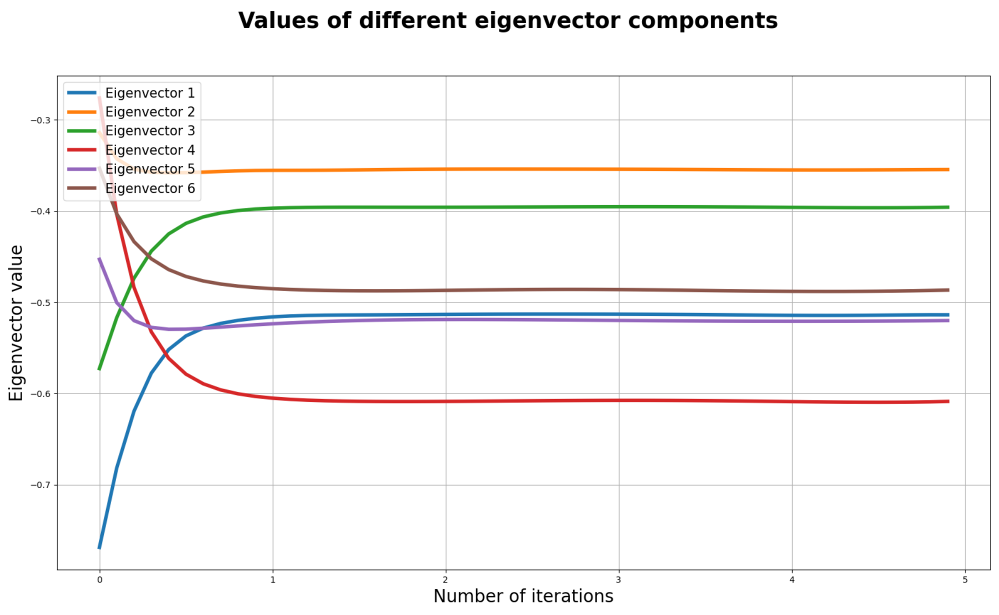

Figure 12: Convergence of the components of the eigenvector corresponding to the largest eigenvalue. The algorithm is stopped once the error of the cost function is below 0.00001.

One can observe from figure 12 that the components of ν_{max} quickly converges towards its correct values. However, the eigenvector components are very sensitive, so many iterations were needed to obtain the maximum eigenvalue (720244 to be exact). The error of the cost function after these iterations were found to be $9.9941 * 10^{-6}$.
We can compare the largest eigenvector and eigenvalue to the one found by numpy with the function numpy.linalg.eig as seen in below.

$$\nu_{max,norm,numpy} = [-0.43040961, -0.78866792, 0.177719, -0.23351243, 0.31853479, -0.07193036] \tag{27}$$

$$\lambda_{max,numpy} = 3.1220782539801926 \tag{28}$$

It is first observed that the maximum eigenvalue produced from the neural network is very close to the maximum eigenvalue produced by the numpy algorithm. However, the eigenvectors are different from one another. This is caused by the maximum eigenvalue having a multiplicity of at least 2 such that the two eigenvectors in equations 25 and 27 are linearly dependent (Axler, 2015).

Exercise e): Discussion

Comparing the analytical, forward Euler and neural network solutions to the heat equation

We first compare the results from the analytical, forward Euler and neural network solutions to the heat equation. The following comparison then assumes the analytical solution is the correct solution to the heat equation. This statement is not entirely correct as there exists some assumptions in the derivation of the solution of the heat equation (between equation 13 and 14). Even still, the analytical solution is deemed the correct solution.
One observes from comparing figures 1, 2 and 3 to figures 4, 5 and 6 that the forward Euler approaches the analytical solution more and more as the step size in space (and therefore also step size in time due to $\frac{\Delta t}{\Delta x^2} < 0.5$) approaches zero. This is due to the forward Euler approach being rooted in the definition of the derivative, meaning that smaller step sizes leads to a better approximation of the true derivative.
The neural network solution is also equivalent to the analytical solution for $t = 0$ (initial heat distribution) and $t = 0.5$. However, the neural network implementation deviates from the analytical solution at $t = 1$. This was a recurring problem with the neural network implementation for later times, as the trained model was unable to reduce the cost function below a certain threshold. Because of this and the fact that the temperature is very low at later times, means that deviations from the analytical solution occurs at later times.
It was shown that both the forward Euler and neural network implementations can solve the one dimensional heat equation, but it is difficult to extrapolate this to further dimensions. However, Koryagin et.al. showed that it is possible to solve the two dimensional heat equation using a similar dense layer neural network as done here (Koryagin et.al., 2019). We also know that the two dimensional heat equation has an analytical solution and can be solved using forward Euler. However, when the system gets more complicated than this, the neural network should still be able to yield an approximate solution where the forward Euler cannot. Such a neural network may be the deep Euler method (DEM) (Xing et.al., 2020). It was shown that the DEM was comparable to the exact analytical solution and also the solution produced by the forward Euler scheme. Additionally, the DEM was shown to not have any step size restrictions as the forward Euler scheme. In fact, the DEM was still comparable to the analytical solution for large step sizes. The DEM even showed robustness when noise was added to the data, strengthening the overall utility of the DEM (Xing et.al., 2020). However, this method have mostly been tested on ordinary differential equations (the heat equation is a partial differential equation), but there may be possibility of utilizing this method on PDEs as well.
Another method proposed by Flament et. al. is able to produce a bundle of solutions for an ODE, meaning that the neural network can solve the ODE with varying initial and boundary conditions. This method could have been useful in this project if the method could be extrapolated to PDEs and if multiple rods with multiple initial conditions, lengths and boundary conditions were considered. This is a very strong tool as ODEs can be solved on the fly by passing them through a network that is already trained. However, Flament et. al. mentions that the results produced from the algorithm should be double checked against other numerical solutions, as the broadness of the method might not be entirely accurate (Flament et. al., 2020).

Discussing the results from the Yi et.al. method of finding the smallest/largest eigenvalues of a matrix

The neural network code, with a few adjustments, was also able to find the smallest/largest eigenvalue of a real, symmetric and randomly generated square matrix using the method proposed by Yi et.al. (Yi et.al., 2004). One can observe that the eigenvalues from my own neural network in equation 26 is almost identical to the one that numpy achieves in equation 28. However, the eigenvectors are different as can be observed from equation 25 and 27. This is due to the multiplicity of the eigenvalue where multiple different eigenvectors can produce an eigenvalue (Axler, 2015).
The paper by Yi et.al. do not cover how to obtain the eigenvalues that are not the smallest/largest eigenvalues, but there are other methods to obtain these. However, it has been proven, both in the paper by Yi et.al. and in this project, that a neural network implementation can find the smallest/largest eigenvalues of a real, symmetric and square matrix.
An extension to the approach used in this project was proposed by Tang and Li where the matrix in question does not have to be symmetric, but skew-symmetric ($A^T = -A$) (Tang and Li, 2009). Additionally, the matrix can be of order 2n instead of n as in this exercise.

Conclusion

It has been shown in this project that both a forward Euler scheme and a neural network algorithm can accurately approximate the analytical solution to the one dimensional heat equation. In particular, it was shown that the forward Euler yields a progressively better approximation to the analytical solution to the heat equation when the step sizes in both time and space decreases. When the step size was sufficiently small, the approximation came arbitrarily close to the analytical solution for all time and space.
The neural network implementation also succeeded to approximate the analytical solution to the heat equation when the number of iterations was large. However, the approximation became slightly off for later times and lower temperatures as the neural network algorithm could not sufficiently reduce the cost function with 1000000 iterations. This was a recurring problem regardless of the input parameters.
The results from the method of finding the smallest/largest eigenvalue of a real, symmetric and square matrix proposed by Yi et.al. was accurately reproduced. The method was applied to a $6x6$ matrix and the solution could become very close to the solution by generic eigenvalue solver (Python package numpy in this case). The method proved to be relatively fast and stable (Yi et.al., 2004).

Future work

This project has shown that a simple neural network implementation can yield an approximate solution to the heat equation and find the smallest/largest eigenvalue of a matrix. However, I do believe that further improvements can be applied in order to reach these solutions faster. Some of these improvements are discussed in project 2 (`https://github.com/sanderwl/FYS-STK4155/tree/master/Project2/Project2`) and include exact optimization of input parameters and activation functions within the hidden layers. However, the results obtained in this exercise was deemed good without the optimizations.

References

- Axler, S., 2015, *Linear Algebra Done Right*, Third Edition, Springer International Publishing, page 136, Available from: `https://zhangyk8.github.io/teaching/file_spring2018/linear_algebra_done_right.pdf`, DOI 10.1007/978-3-319-11080-6

- Flament, C., Protopapas, P., Sondak, D., 2020, *Solving Differential Equations Using Neural Network Solution Bundles*, Available from `https://arxiv.org/abs/2006.14372`, arXiv:2006.14372v1

- Hancock, M. J., 2006, *The 1-D Heat Equation, Linear Partial Differential Equations*, page 5, Available from `https://ocw.mit.edu/courses/mathematics/18-303-linear-partial-differential-equations-fall-2006/lecture-notes/heateqni.pdf`

- Kingma, D. P., Ba, J. L., 2014 *Adam: A Method for Stochastic Optimization*, page 1, Available from: `https://arxiv.org/pdf/1412.6980.pdf`, arXiv:1412.6980v9

- Koryagin, A., Khudorozkov, R., Tsimfer., S., 2019, *PYDENS: A PYTHON-FRAMEWORK FOR SOLVING DIFFERENTIAL EQUATIONS WITH NEURAL NETWORKS*, Available from: `https://arxiv.org/pdf/1909.11544.pdf`

- Tang, Y., Li, J., 2009, *Another neural network based approach for computing eigenvalues and eigenvectors of real skew-symmetric matrices*, Elsevier Ltd, Available from: `https://core.ac.uk/download/pdf/82245574.pdf`

- Venkanna, B.K, 2010, *Fundamentals of Heat and Mass Transfer*, New Delhi PHI Learning, page 38, ISBN-978-81-203-4031-2

- Xing, S., Cheng, X., Liang, K., 2020, *Deep Euler method: solving ODEs by approximating the local truncation error of the Euler method*, Available from: `https://www.researchgate.net/publication/340180255_Deep_Euler_method_solving_ODEs_by_approximating_the_local_truncation_error_of_the_Euler_method`

- Yi, Z., Fu, Y., Tang, H.J., 2004, *Neural Networks Based Approach Computing Eigenvectors and Eigenvalues of Symmetric Matrix*, Available from: `https://www.sciencedirect.com/science/article/pii/S0898122104901101`