Solving differential equations with neural networks

Ivar Lønning
(Dated: December 20, 2022)

We discuss solution methods of differential equations, specifically the forward Euler method and a method that is based on neural networks. We solve the diffusion equation numerically and compare the results to the analytical solution. The forward Euler scheme gives a smaller relative error than the neural network method, but both methods give reasonable looking, exponentially decaying solutions when compared to the analytical solution. We also estimate an eigenvector and eigenvalue of a symmetric matrix by solving a system of ordinary differential equations using the neural network method. The relative error in the eigenvalue is 0.007, which demonstrates that the method works. Further improvement on the method is necessary. (The code used is available at https://github.com/ivarlon/fys-stk4155/tree/main/project3.)

I. INTRODUCTION

There exist many methods to solve physical problems, both analytically and numerically, and new methods are constantly being researched. One domain of problems is differential equations (DEs), which are ubiquitous in physics and in the sciences in general. In some cases, DEs can be solved analytically, but often they require a numerical solution method. A common numerical method is the forward Euler scheme, which can be used to solve ordinary differential equations (ODEs) as well as partial differential equations (PDEs). In recent decades, there have also been methods developed that use neural networks (NNs) to solve DEs. The basic structure of the method is to propose a trial solution that depends on the output from the NN and respects the initial and boundary conditions of the problem, and then to train the NN to make the trial solution correspond to the relevant DE. An interesting application is to use the method to find eigenvalues and eigenvectors of a symmetric matrix, by solving a system of ODEs.

In this report we investigate the forward Euler method as well as the NN method for solving DEs. We will be solving the diffusion equation, which is a PDE. In addition we will apply the NN method to find eigenvectors of a matrix. Our work is primarily a proof-of-concept for the NN method, in that we wish to demonstrate that it works, but the accuracy and usefulness of the method is a matter of improvement and further research. In section II we discuss the diffusion equation and its solution, as well as describe the forward Euler method and the NN method. The results of the various methods are presented and discussed in section III. We give a summary and remarks on future improvement in section IV.

II. METHODS

The diffusion equation

A partial differential equation (PDE) which often appears in physics is the diffusion equation, which in one dimension is

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad t \geq 0, x \in [0, L]. \tag{1}$$

Here $u(x,t)$ is the unknown function we wish to find. According to this equation, u will increase in regions where it is convex, and decrease where it is concave, hence the equation can be used to describe processes such as e.g. heat diffusion on a rod. In our current case u will have initial conditions $u(x, t = 0) = \sin(\pi x)$, and have Dirichlet boundary conditions $u(x = 0, t) = u(x = L, t) = 0$. We will also set $L = 1$. With these conditions we may find an analytical solution to the problem. We write an ansatz solution as a product of two functions, one being a function of x and the other of t:

$$u(x,t) = g(x)h(t). \tag{2}$$

Then if we differentiate with respect to x and t, and insert into eq. (1), we get

$$g(x)\frac{dh(t)}{dt} = h(t)\frac{d^2 g(x)}{dx^2}. \tag{3}$$

We may now divide by $g(x)h(t)$, which leaves the left-hand side only dependent on t and the right-hand side only on x, meaning that these expressions must be equal to a constant k^2:

$$\frac{1}{h(t)}\frac{dh(t)}{dt} = \frac{1}{g(x)}\frac{d^2 g(x)}{dx^2} \equiv k^2. \tag{4}$$

This gives us two simple ODEs for g and h, with general solutions

$$h(t) = e^{k^2 t}, \quad g(x) = g_1 e^{kx} + g_2 e^{-kx}, \tag{5}$$

with g_1 and g_2 being constants. In order for these solutions to fulfill the initial and boundary conditions, we must have

$$g_1 = -g_2 = \frac{1}{2} \quad \text{and} \quad k^2 = -\pi^2, \tag{6}$$

so the final solution is

$$\begin{aligned} u(x,t) = g(x)h(t) &= \frac{e^{i\pi x} - e^{-i\pi x}}{2} e^{-\pi^2 t} \\ &= \sin(\pi x) e^{-\pi^2 t}. \end{aligned} \tag{7}$$

Numerical solution methods

The diffusion equation may also be solved numerically, which may be the only option for some initial and boundary conditions. We will first describe a traditional solution method, an explicit forward Euler algorithm (see [2] for more). Then we will describe a solution method using neural networks.

The explicit forward Euler method is based on discretising space and time respectively into N_x and N_t points, as follows:

$$x \to i\Delta x, \quad t \to j\Delta t, \tag{8}$$

$$i = 0, 1, \ldots, N_x - 1, \quad j = 0, 1, \ldots, N_t - 1$$

where Δx is the spatial step length and Δt the time step length. Furthermore, the solution is also discretised as

$$u(x,t) \to u(x_i, t_j) \equiv u_i^j. \tag{9}$$

The method uses a forward difference scheme for the time derivative:

$$\frac{\partial u}{\partial t} \approx \frac{u_i^{j+1} - u_i^j}{\Delta t}, \tag{10}$$

and a centered difference scheme for the second spatial derivative:

$$\frac{\partial^2 u}{\partial t^2} \approx \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2}. \tag{11}$$

The discretised version of the diffusion equation (1) then becomes

$$\frac{u_i^{j+1} - u_i^j}{\Delta t} = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2}, \tag{12}$$

which can be solved to give an explicit formula for the solution for the next timestep:

$$u_i^{j+1} = u_i^j + \frac{\Delta t}{\Delta x^2}\big[u_{i+1}^j - 2u_i^j + u_{i-1}^j\big]. \tag{13}$$

If we arrange values for the inner points, i.e. those with index $i = 1, \ldots, N_x - 2$, into a vector $\mathbf{u}^j = [u_1^j, \ldots, u_{N_x-2}^j]^T$, we may write the above equation as a matrix-vector equation.

$$\mathbf{u}^{j+1} = A\mathbf{u}^j, \tag{14}$$

where A is an $(N_x - 1) \times (N_x - 1)$ symmetric matrix with 2 on the diagonal and -1 on the super- and subdiagonal. Iterating this equation is how we solve the PDE. The accuracy of the numerical solution depends on the time and step length. A criterion for the stability of the numerical solution is that $\Delta t/\Delta x^2 \leq 1/2$.

We may also use neural networks (NNs) to solve the PDE, motivated by the fact that they are universal function approximators (for a good overview of NNs and applications, see [1]). We write a trial solution of the PDE as

$$u_{\text{tr}} = h_1(x,t) + h_2(x,t,\text{NN}(x,t)), \tag{15}$$

where $h_1(x,t)$ is a function that guarantees that the trial solution fulfill the initial conditions, and $h_2(x,t,\text{NN})$ is the part of the solution that depends on the NN. Additionally, u_{tr} must respect the boundary conditions. Therefore, we may specify h_1 and h_2 as

$$h_1(x,t) = (1-t)u(x,t=0) \tag{16}$$

$$h_2(x,t,\text{NN}) = x(1-x)t\text{NN}(x,t),$$

which ensures the initial and boundary conditions are met. We have described NNs in an earlier project, and therefore we are content here with merely summarising the basics of NNs. Our NN is a function which takes input (x,t) and through a series of L transformations gives an output $\text{NN}(x,t)$. Each transformation is of the form

$$\mathbf{a}^l = \sigma\Big(P^l \begin{bmatrix} \mathbf{a}^{l-1} \\ 1 \end{bmatrix}\Big), \tag{17}$$

where $\mathbf{a}^l$ is the l-th transformation (with $\mathbf{a}^0 = (x,t)$), $\sigma(\cdot)$ is a non-linear activation function (applied element-wise), and P^l is a matrix containing weights and biases. A commonly used activation function is the sigmoid:

$$\sigma(x) = \text{sigmoid}(x) = \frac{1}{1+e^{-x}}. \tag{18}$$

The NN is therefore specified by its parameters P^l (and by the activation functions used). The optimal parameters are such that u_{tr} solves the PDE eq. (1), i.e.

$$\frac{\partial u_{\text{tr}}}{\partial t} - \frac{\partial^2 u_{\text{tr}}}{\partial x^2} = 0. \tag{19}$$

If we define the cost function

$$C(x,t,\text{NN}(x,t)) = \Big[\frac{\partial u_{\text{tr}}}{\partial t} - \frac{\partial^2 u_{\text{tr}}}{\partial x^2}\Big]^2, \tag{20}$$

the NN model we want is the one that minimises this cost function. The optimal parameters can thus be found by the gradient descent algorithm, which minimises the cost function:

$$P^l \leftarrow P^l - \eta \nabla_{P^l} C(x,t,\text{NN}(x,t)), \tag{21}$$

which is iterated until the NN gives a desired output. Here, η is the learning rate which specifies the degree to which the parameters are changed for each iteration, and $\nabla_{P^l} C(x,t,\text{NN}(x,t))$ is the gradient of the cost function with respect to the parameters P^l.

Using NNs to find eigenvectors of a symmetric matrix

As NNs can be used to solve PDEs, *a fortiori* they can be used solve ordinary differential equations (ODEs).

One application is to find the eigenvectors of a symmetric matrix. This was investigated by Yi *et al.* in [3]. We define the following system of ODEs:

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

where $\mathbf{x}(t) \in \mathbb{R}^n$ and

$$f(\mathbf{x}) = \left[\mathbf{x}^T\mathbf{x}A + (1 - \mathbf{x}^T A\mathbf{x})\right]\mathbf{x}. \tag{23}$$

Here A is an $n \times n$ symmetric matrix whose eigenvectors we wish to find. Notice that if $\mathbf{x}$ is an eigenvector of A, it is also a fixed point of f, i.e. $f(\mathbf{x}) = \mathbf{x}$. Then the right hand side of eq. (22) is zero, and we have a stationary state. It is proven in [3] that any non-zero initial condition $\mathbf{x}(t = 0) = \mathbf{x}_0$ will converge in time t to an eigenvector of A. By solving the ODE with a NN, we can therefore extract an eigenvector of A by evaluating the solution at late times. We suggest the following trial solution:

$$\mathbf{x}_{\rm tr}(t) = (1 - t)\mathbf{x}_0 + t\mathbf{NN}(t), \tag{24}$$

where $\mathbf{NN}(t)$ is the output of the NN. The NN is trained by minimising a cost function

$$C(t, \mathbf{NN}(t)) = \left[\frac{d\mathbf{x}_{\rm tr}(t)}{dt} + \mathbf{x}_{\rm tr}(t) - f(\mathbf{x}_{\rm tr}(t))\right]^2. \tag{25}$$

Once we have estimated a solution and extracted an estimated eigenvector $\tilde{\mathbf{v}}$, we may find the corresponding eigenvalue $\tilde{\lambda}$ by $\tilde{\lambda} = \frac{\tilde{\mathbf{v}}^T A\tilde{\mathbf{v}}}{\tilde{\mathbf{v}}^T\tilde{\mathbf{v}}}$.

It's worth pointing out that the work by Yi *et al.* was focussed on recurrent NNs, whereas our treatment has been centered around feed forward NNs. Our methodology therefore differs in important respects from that of Yi *et al.*. However, both approaches have in common the basic premise of solving an ODE to get the eigenvectors of a symmetric matrix, which relies on mathematical theory which is proved in their paper.

III. RESULTS AND DISCUSSION

We wish to make an initial comment on our implementation of the methods above. We wrote our code in Python. There exist good frameworks in Python for NNs: e.g. the Tensorflow library. Unfortunately, we were unable to make this work on the operating system we used, *viz.* Windows 7. We therefore wrote our own simple, bare-bones code specifically designed to solve the relevant problems.

We solved the diffusion equation with the explicit forward Euler scheme for two different spatial step sizes $\Delta x = 1/10, 1/100$. In order to ensure the stability of the solution we chose a timestep $\Delta t = \Delta x^2/2$. We show the solution for times $t = 0.1, 0.5$ in figs. 1 and 2 for the two step sizes respectively, along with the analytical solution for the same times. The shape is approximately the same, disregarding the obvious jagged nature of the solution for the larger stepsize. However, a small deviation from the analytical solution is visible for the larger stepsize at $t = 0.1$. For this stepsize, we get a mean relative error $\epsilon = 0.012$ at $t = 0.1$ and $\epsilon = 0.057$ for $t = 0.5$. For the smaller stepsize, we get $\epsilon = 0.0002$ and $\epsilon = 0.0008$, for the two times respectively. So the larger step size gives an error of more than 5% for late times, and it may therefore be worthwhile to decrease the step size. Nevertheless, even a fairly rough discretisation with $\Delta x = 1/10$ gives a decent solution of the diffusion equation. An advantage of the method is that the solution may be calculated quickly and efficiently, due to the matrix A being sparse and having little memory overhead.

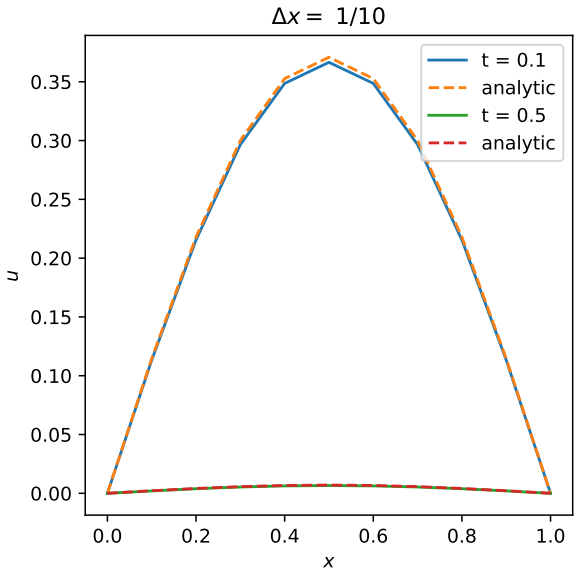

FIG. 1. This shows the forward Euler solution with $\Delta x = 1/10$ for two different times, along with the analytic solution for the same times.

We also solved the diffusion equation with the NN method. For our case, we discretised time and space into 10 points each. The NN contained $L = 3$ layers, with two hidden layers and one output layer. The first hidden layer was of size 100, and the second of size 25; the weights and bias parameters were initialised from a standard normal distribution. The NN used the sigmoid as activation function for the hidden layers, and no activation function for the output layer. We trained the NN using stochastic gradient descent, with 25 epochs and a batch size of 4. The convergence of the gradient descent algorithm was very sensitive to the learning rate, and we used a learning rate of 0.01. We experimented with using momentum in the gradient descent, but that worsened the convergence, and so we ended up using no momentum. The solution $u_{\rm tr}$ is shown in fig. 3, with the analytical solution u (eq. (7)) in fig. 4 for reference. The two plots clearly resemble each other, both showing

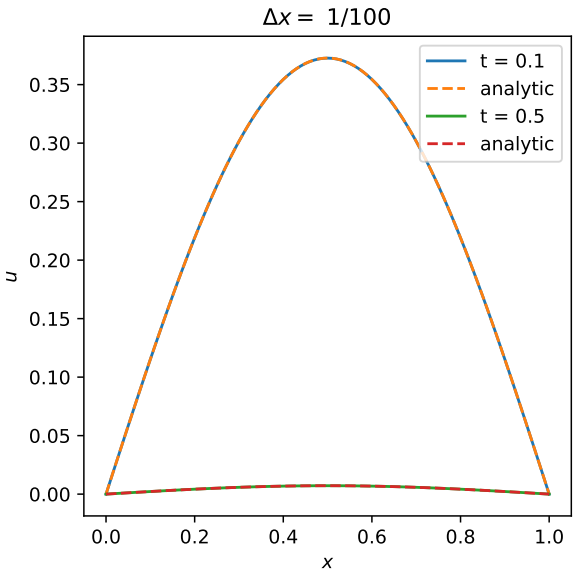

FIG. 2. This shows the forward Euler solution with $\Delta x = 1/100$ for two different times, along with the analytic solution for the same times.

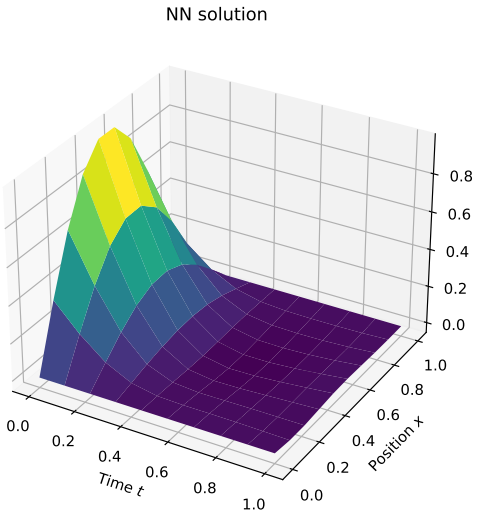

FIG. 3. This shows the solution of the diffusion equation using a NN.

exponential decay over time. However, the NN solution doesn't decay as strongly as the analytical solution. This is shown e.g. by the difference $u_{\text{tr}} - u$ in fig. 5, where we see a clear deviation in the center regions especially at early times. The relative error was calculated, excluding the initial solution at $t = 0$ and the boundaries, where the error is zero by construction. The mean relative error was found to be 0.0706, with a maximum relative error

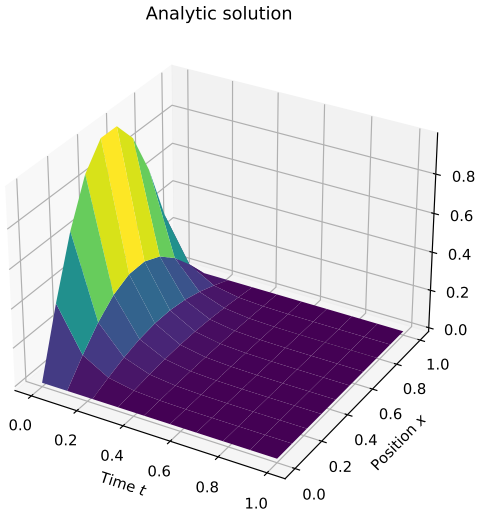

FIG. 4. This shows the analytical solution of the diffusion equation.

0.2863 and a minimum 0.0007. This is therefore a fairly rough solution. Its mean error is larger than the error the forward Euler scheme makes even with a similar spatial resolution. Using NNs to solve PDEs therefore seems to be an inferior choice to some traditional methods. However, the NN solution may be improved by training the NN for longer and by tweaking hyperparameters such as the NN architecture and the learning rate.

The NN method was used to obtain eigenvectors and eigenvalues of a 6×6 symmetric matrix A, generated randomly from a standard normal distribution:

$$A = \begin{bmatrix} 1.764 & 0.675 & 0.87 & 1.277 & 2.069 & -0.411 \\ 0.675 & -0.151 & 0.009 & -0.222 & -0.655 & 0.916 \\ 0.87 & 0.009 & 0.444 & -1.11 & 0.77 & -0.546 \\ 1.277 & -0.222 & -1.11 & 0.654 & 0.339 & -1.361 \\ 2.069 & -0.655 & 0.77 & 0.339 & 1.533 & 0.561 \\ -0.411 & 0.916 & -0.546 & -1.361 & 0.561 & 0.156 \end{bmatrix}$$

The NN was described by two hidden layers, with 4 and 8 nodes respectively, and with the sigmoid as activation function. We trained the network on 200 time samples $t \in [0, 1]$, using a stochastic gradient descent with batch size 20 and 2000 epochs and a learning rate of 0.001. The initial value $\mathbf{x}_0$ was gotten from a standard normal distribution. After training we evaluated the solution at $t = 1$ to get an estimated eigenvector. However, we observed that not every $\mathbf{x}_0$ resulted in convergence of the method, which was discouraging. But there were some initial values that gave good results. One of these was

$$\mathbf{x}_0 = [1.230, 1.202, -0.387, -0.302, -1.049, -1.420]^T.$$

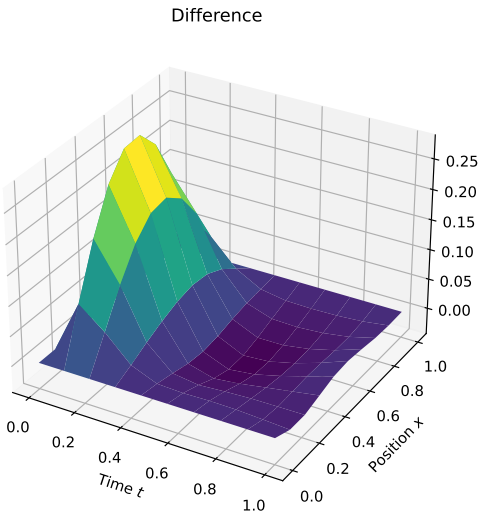

FIG. 5. The difference between the NN solution and the analytic solution.

We then obtained an estimated eigenvector

$$\tilde{\mathbf{v}} = [-0.377, -0.548, -0.371, 0.059, 0.577, 0.289]^T, \quad (26)$$

with a corresponding eigenvalue $\tilde{\lambda} = 0.589$. (Note that this solution has been normalised.) We can compare to the actual eigenvector gotten by traditional methods:

$$\mathbf{v} = [-0.360, -0.567, -0.406, 0.052, 0.573, 0.231]^T, \quad (27)$$

with corresponding eigenvalue $\lambda = 0.593$. We see that the estimate is not quite exact, but the error in the estimated eigenvector is

$$(\mathbf{v} - \tilde{\mathbf{v}})^T(\mathbf{v} - \tilde{\mathbf{v}}) = 0.073, \quad (28)$$

and the relative error of the eigenvalue is 0.007. This demonstrates that we have found a decent approximation and thus that this method works. (The probability of *randomly* generating a 6 dimensional vector this close to an actual eigenvector, is practically zero.) Because the method we have used didn't give convergence for any non-zero initial condition, leading to a blind search for a workable initial condition, it is desirable to further improve the method. One potential avenue is to train on larger times than $t = 1$, up to $t = T > 1$, which may lead to stationary states not reachable within $t \leq 1$. One may then correspondingly tweak the expression for the trial solution (24) by having the initial condition vanish not at $t = 1$ but at $t = T$. One may also train the network on a larger sample, and for more epochs. Finally, we may wish to alter the initial state so that we have greater control over which eigenvector the method converges to, perhaps by making it orthogonal to already known eigenvectors.

To summarise our results, we conclude that the NN method gives inferior results compared to traditional methods to solve differential equations (DEs). Why not just use forward Euler instead? An advantage of the NN method is the generality it has in being able to solve DEs by simply using the (squared) DE as cost function, meanwhile the forward Euler equations will look different for different DEs. But the training process of a NN requires a lot of computation, and it seems superfluous given the existence of more accurate methods. A downside of the forward Euler scheme is the requirement that $\Delta t \leq \Delta x^2/2$, meaning that we are forced to compute the solution for a lot of time iterations if we increase the spatial resolution. But there exist other methods that don't have this requirement, e.g. the implicit Crank-Nicolson scheme. Which solution method is best is obviously a problem-specific question and also depends on the desired accuracy. But our results demonstrate that the NN approach works, and given enough training and suitable hyperparameters it may give a satisfactory solution.

IV. CONCLUSION

We investigated numerical methods for solving differential equations, which included the traditional explicit forward Euler scheme as well as a method using neural networks. We solved the one dimensional diffusion equation using both methods. For the given initial and boundary conditions we could calculate the exact analytical solution, which could be used for comparison with the performance of the numerical methods. The forward Euler scheme gave very small errors, especially with a spatial resolution $\Delta x = 1/100$. In contrast, the solution obtained by the NN method had a significant error especially in the center. However, the solution had a similar shape to the exact solution and this indicated that the method works in principle. We also looked at solving a system of ODEs in order to find the eigenvectors and -values of a symmetric matrix. Using the NN method we were able to find an eigenvector and eigenvalue, with a relative error in the eigenvalue of 0.007, which nicely demonstrated the power of the method. The method is sensitive to the initial value, and after training of the NN the trial solution was not always observed to converge to an eigenvector. The method may be further improved by training on larger timescales and simply by training on larger samples and for more epochs.

An additional area of inquiry is in quantification of the uncertainty of the solution gotten by the NN method. The error made using finite difference schemes such as forward Euler can be estimated/bounded by well established theory. We would like to obtain a similar error estimate/bound on the NN solution, and this question could be further investigated.

[1] Ian Goodfellow, Yoshua Bengio & Aaron Courville (2015). *Deep Learning*, MIT Press.
[2] Morten Hjorth-Jensen (2015). *Computational Physics, Lecture Notes Fall 2015*. Available online at https://raw.githubusercontent.com/CompPhysics/ComputationalPhysics/master/doc/Lectures/lectures2015.pdf
[3] Zhang Yi & Yan Fu (2003). "Neural networks based approach for computing eigenvectors and eigenvalues of symmetric matrix" in *Computers and Mathematics with Applications* 47 (2004), pp. 1155-1164.