Solving partial differential equations with neural networks

FYS-STK4155: Project 3

Morten Ledum, Håkon Kristiansen & Kari Eriksen
github.com/mortele/FYS-STK4155

December 17, 2018

Abstract

We solve the heat equation in one spatial dimension with direchlet boundary conditions and a sinusoidal initial condition. A number of different solution schemes are explored: Finite difference variants *explicit* and *implicit* Euler, and the Crank-Nicholson scheme, in addition to a finite element scheme using P2 piecewise Lagrange interpolating polynomial basis functions. Lastly, a neural network is designed which is able to solve the same problem. Comparison between the methods reveals that the neural network approach is not useful in the solving of this specific problem (in terms of cost effectiveness). Although the neural network solution is superior in the case of very small discretized lattice sizes and a very small number of time steps employed, the error scaling of the traditional methods prove to be far superior.

Contents

INTRODUCTION

Solving partial differential equations (PDE) can be a challenge in the world of applied mathematics. They describe everything from the Black-Scholes model in finance to the Schrödinger equation in the field of quantum mechanics. In many cases there are no analytical solutions, or they are time-consuming and complex to solve. One therefore often turns to numerical solutions and these methods come in a wide range of variations. Finite difference is a well-known and widely used method that approximates a differential eqution with a discretized version using Taylor expansion. These are often easy to implement, however some of the methods suffer from weak stability for some parameters. We therefore consider a new approach to the problem of solving differential equations numerically.

Machine learning (ML) is a field in computational science growing in a fast pace and is continuously applied to new problems. With the enormous amount of data in the world today the need for automated methods of data analysis is great, which is what ML originated from. But the interest for ML has developed to other fields and solving PDE with neural network has been one of the new applications of ML.

In this project we will therefore take a closer look at deep neural networks as a method for solving PDE. We study the diffusion equation and solve it using different methods. First we employ several finite difference methods; explicit and implicit euler scheme, the Crank-Nicolson scheme. Then we study finite element methods, Morten skriver du noe her? Before we move on to training a neural network to the solution of the diffusion equation. As much of the theory on neural network has been discussed in a previous report, project 2, we will not dwell on this subject. But solving differential equations with NN goes about somewhat different than regression or classification analysis with NN. Thus we discuss it to some extent in the theory, where discription of the finite difference methods also can be found together with finite element methods. And a derivation of the analytical solution is of course given.

THEORY

The heat equation

The heat equation is a partial differential equation (in space x and time t) which describes the evolution of temperature differences in a region of space over a time interval. It is based on *Fourier's law*; the rate of heat flow through a surface is proportional to the temperature gradient across the surface, i.e.

$$\mathbf{q} = -k\nabla T = -k\frac{\partial T}{\partial x}, \tag{1}$$

where $\mathbf{q}$ denotes the heat flux density, k is the thermal conductivity of the surface material, and T represents the temperature. Changes in temperature are proportional to changes in internal energy, with the proportionality constant being the specific heat capacity c_p. With the arbitrary energy zero point placed at absolute zero, this can be written as

$$Q = c_p \rho T, \tag{2}$$

with Q being the internal energy and ρ denoting the mass density. This is essentially just a restatement of (a shifted) *first law of thermodynamics*, in the absence of applied work. The total heat energy contained in a region $[a, b]$ is given by the integral

$$\int_a^b \mathrm{d}x\, c_p \rho T(x,t). \tag{3}$$

Integrating over a small region of space and considering the change in internal energy over a short time interval (assuming c_p and ρ are both time-independent and spatially homogenous) gives

$$\begin{aligned}\Delta Q =& c_p\rho \int_x^{x+\Delta x} \mathrm{d}\chi \left[T(\chi, t+\Delta t) - T(\chi,t)\right] \\ =& c_p\rho \int_x^{x+\Delta x} \mathrm{d}\chi \int_t^{t+\Delta t} \mathrm{d}\tau\, \frac{\partial T}{\partial \tau}.\end{aligned} \tag{4}$$

Over a short time period Δt, the change in internal energy of a short segment of length Δx must be entirely due to the heat flux in/out of the boundaries,

$$\begin{aligned}\Delta Q &= k\int_t^{t+\Delta t} \mathrm{d}\tau \left[\frac{\partial T(x+\Delta x,\tau)}{\partial x} - \frac{\partial T(x,\tau)}{\partial x}\right] \\ &= k\int_t^{t+\Delta t} \mathrm{d}\tau \int_x^{x+\Delta x} \mathrm{d}\chi\, \frac{\partial^2 T}{\partial \chi^2}.\end{aligned} \tag{5}$$

By conservation of energy, the difference between Eq. (4) and Eq. (5) must obviously vanish. Since we are integrating over the same spatial and temporal regions in both equations, this means that the integrand must vanish identically:

$$\frac{k}{c_p\rho}\frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t}. \tag{6}$$

This is known as the *heat equation* and is a special case of the more general diffusion equation.

Closed form solution

The 1D heat equation may be solved by applying a separation of variables ansatz, i.e. we assume the solution $u(x,t)$ takes the form

$$u(x,t) = X(x)T(t), \tag{7}$$

with $X(x)$ carrying all the x-dependence, and $T(t)$ carrying the corresponding time dependence. Introducing the compact notation

$$u_x \equiv \partial_x u = \frac{\partial u}{\partial x}, \quad \text{and} \quad u_t \equiv \partial_t u = \frac{\partial u}{\partial t}, \tag{8}$$

we find by insertion of the ansatz into Eq. (6):

$$\begin{aligned}
\alpha^2\, u_{xx} &= u_t \\
\alpha^2\, \partial_x\Big[\partial_x X(x)T(t)\Big] &= \partial_t X(x)T(t) \\
\alpha^2\, \partial_x\Big[X_x T + X T_x\Big] &= X_t T + X T_t \\
\alpha^2\, \Big[X_{xx} T + 2X_x T_x + X T_{xx}\Big] &= X T_t \\
\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} &= \frac{1}{\alpha^2\, T(t)}\frac{\partial T(t)}{\partial t},
\end{aligned} \tag{9}$$

where we defined $\alpha^2 \equiv k/c_p\rho$ and used the fact that $X_t = T_x = 0$. As the left hand side is independent of t and the right hand side is independent of x, the equality can only be achieved if both sides are constant. This reduces the original partial differential equation into a set of two ordinary differential equations

$$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = k \tag{10}$$

$$\frac{1}{\alpha^2\, T(t)}\frac{\partial T(t)}{\partial t} = k, \tag{11}$$

with k an undetermined constant.

Depending on the value of k, the spatial part has solutions $X(x) = Ax + B$ (if $k = 0$), $X(x) = Ae^{\mu x} + Be^{-\mu x}$ (if $k = \mu^2 > 0$), or $X(x) = Ae^{i\mu x} + Be^{-i\mu x}$ (if $k = -\mu^2 < 0$). The temporal equation has solutions $T(t) = C$ (if $k = 0$), or $T(t) = Ce^{\alpha^2\mu^2 x}$ (otherwise).

Applying the boundary conditons

In order to make progress, we need to apply the specific boundary and initial conditions. In our case, the boundaries at $x = 0$ and $x = L = 1$ vanish, and the initial spatial solution takes the form $u(x, t = 0) = \sin \pi x$. If the k constant of Eq. (10) vanishes, then both constants A and B vanish due to the boundary conditions. The same is true if $k = \mu^2 > 0$. It follows that the only non-trivial solutions arise when $k = -\mu^2 < 0$, in which case we find (left boundary)

$$X(0) = A\cos\mu x + B\sin\mu x = 0 \Rightarrow A = 0$$

and (right boundary)

$$X(1) = B\sin\mu x = 0 \Rightarrow \mu = \pi n.$$

This gives rise to an infinite set of equations—one for each n—which determine the Fourier coefficients of the initial condition $u(x, t = 0)$:

$$u(x, t = 0) = \sum_{n=1}^{\infty} B_n \sin(n\pi x), \tag{12}$$

with

$$B_n = 2\int_0^1 \mathrm{d}x\, u(x, t = 0)\, \sin(n\pi x). \tag{13}$$

The temporal equation, Eq. (11), can now be solved only applying the boundary conditions. With the value of μ fixed at $\mu = n\pi$, we obtain

$$T(t) = e^{-n^2\pi^2\alpha^2 t}. \tag{14}$$

Applying the initial condition

Combining the $X(x)$ and $T(t)$ solutions we obtain a series representation of the solution in terms of the Fourier coefficients of the initial condition $u(x, t = 0)$, in which the higher frequency modes of the initial solution decays more rapidly than the corresponding lower frequency modes,

$$u(x,t) = \sum_{n=1}^{\infty} B_n \sin(n\pi x)\, e^{-n^2\pi^2\alpha^2 t}. \tag{15}$$

It is easy to see that the steady-state solution is achieved when $u(x,t) = 0$, since both boundaries (left and right) act as sinks for the initial heat energy contained in the system, and no heat is ever *added*. In our case, the initial condition makes the general solution Eq. (15) take on a very simple form. Applying $u(x, t = 0) = \sin \pi x$, it is trivial to evaluate

$$B_n = 2 \int_0^1 \mathrm{d}x \, \sin(\pi x) \, \sin(n\pi x) = \delta_{1n}, \quad (16)$$

due to the orthogonality of $\sin(n\pi x)$ and $\sin(m\pi x)$. This means $B_1 = 1$ and $B_2, B_3, \cdots = 0$, and the full solution to Eq. (6) is given by

$$u(x,t) = \sin(\pi x) \, \mathrm{e}^{-\pi^2 \alpha^2 t}. \quad (17)$$

Explicit scheme

One of the more simple methods in the world of solving ordinary and partial differential equation numerically is the explicit scheme. This is a method that bases itself upon using the previous value of the derivatives in order to find the new ones. We will here derive the explicit scheme for the diffusion equation, Eq. (18).

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \quad (18)$$

We begin with Taylor expantion of $u(x,t)$ in t.

$$u(x, t+\Delta t) = u(x,t) + \Delta t u'(x,t) + \frac{1}{2}\Delta t^2 u''(x,t) + ...$$

Truncating the series after the first term we get an expression of the first derivative of u wrt. t with the truncation error $O(\Delta t)$.

$$u'(x,t) = \frac{u(x, t+\Delta t) - u(x,t)}{\Delta t} + O(\Delta t)$$

This can be simplyfied if we rewrite the derivative of u wrt. t as u_t and discretize.

$$\begin{aligned} u_t &\approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} \\ &\approx \frac{u_{i,j+1} - u_{i,j}}{\Delta t} \end{aligned}$$

This is called the forward Euler method, we move one step forward in time in the integration process using the previous step. This is the simplest version of the explicit scheme. Now we carry out the same procedure for the spatial part.

$$\begin{aligned} u(x + \Delta x, t) &= u(x,t) + \Delta x u'(x,t) + \frac{1}{2}\Delta x^2 u''(x,t) \\ &\quad + \frac{1}{3!}\Delta x^3 u'''(x,t) + ... \\ u(x - \Delta x, t) &= u(x,t) - \Delta x u'(x,t) + \frac{1}{2}\Delta x^2 u''(x,t) \\ &\quad - \frac{1}{3!}\Delta x^3 u'''(x,t) + ... \end{aligned}$$

We truncate at second term in order to get the second order derivative and rewrite to simplify as before.

$$\begin{aligned} u_{xx} &\approx \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2} \\ &\approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} \end{aligned}$$

This is a centered difference method as we are using both previous and latter values in space to find the new second derivative of the spatial coordinate. Adding the two solutions into the original differential equation we get the following.

$$\begin{aligned} u_t &= u_{xx} \\ u_{i,j+1} - u_{i,j} &= u_{i+1,j} - 2u_{i,j} + u_{i-1,j} \frac{\Delta t}{\Delta x^2} \end{aligned}$$

The full diffusion equation on discretized form can further be rewritten by putting $\beta = \Delta t / \Delta x^2$.

$$\begin{aligned} u_{i,j+1} &= \beta(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) + u_{i,j} \\ &= \beta(u_{i+1,j} + u_{i-1,j}) + (1 - 2\beta)u_{i,j} \end{aligned}$$

Finite difference method

The most straightforward way to solve Eq. (6) numerically is through *finite difference methods*, i.e. discretizing time and space on a grid and Taylor expanding the solution to obtain algebraic equation sets. A multitude of different strategies and schemes exists, but we will consider the *explicit forward Euler* scheme.

The spatial region $[0, L]$ is discretized by splitting it into N segments, and considering only the functional values on the points $x_i = i\Delta x$ with $i \in [0, N-1]$. We denote a function $f(x,t)$ evaluated at x_i (and at time t) by $f_i^t = f(x_i, t)$. Considering N spatial points gives a step size Δx between each point of

$$\Delta x = \frac{L}{N-1}. \quad (19)$$

In addition, we introduce a discretization in the temporal dimension with step size Δt.

Let us consider the Taylor expansion of a function $f(x,t)$ considered at fixed t, $f(x;t)$, around a spatial point x. We use the shorthand $h \equiv (x-a)$, and consider the expansion at a point $a \neq x$. The expansions of $f(x+h;t)$ and $f(x-h;t)$ are given by,

$$f(x+h) \approx f(x) + hf'(x) + h^2 f''(x), \quad (20)$$
$$f(x-h) \approx f(x) - hf'(x) + h^2 f''(x), \quad (21)$$

respectively. As t is considered a fixed parameter for the moment, we supressed the second functional argument for notational brevity. The shorthand $f'(x)$ is here used to denote differentiation w.r.t. x. Note that both equations hold with equality if an overall error term proportional to h^3 is added on the right hand side, i.e. $\mathcal{O}(h^3)$. Adding Eqs. (20) and (21) and dividing through by h^2 yields

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \mathcal{O}(h^4)$$
$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \mathcal{O}(h^2), \quad (22)$$

which is the three-point central difference approximation to the second derivative. Note that the resulting error is proportional to h^2 because the third order contributions from Eqs. (20) and (21)—$h^3 f'''(x+h)$ and $-h^3 f'''(x+h)$—cancel exactly. A corresponding temporal first order derivative approximation may be found by simply considering Eq. (20), and considering h to be a small temporal step, x to be a fixed parameter, and varying t. This gives

$$f'(t) = \frac{f(t+h) - f(t)}{h} + \mathcal{O}(h), \quad (23)$$

where the error term is proportional to h^2 when disregarding the last term on the right hand side of Eq. (20), which gives $\mathcal{O}(h)$ after dividing through by h.

Let us now consider Eq. (22) on the previously defined grid, and take $h = \Delta x$. In the Eq. (23) case, we define $h = \Delta t$, and equate $\alpha^2 f''(x)$ with $f'(t)$ as in the heat equation Eq. (6). The result can be solved for $f(x, t+\Delta t)$, i.e. the *next* time step given that the previous step is known:

$$\frac{f_i^{j+1} - f_i^j}{\Delta t} = \alpha^2 \frac{f_{i+1}^j - 2f_i^j + f_{i-1}^j}{\Delta x^2}$$
$$f_i^{j+1} = f_i^j + \beta\Big[f_{i+1}^j - 2f_i^j + f_{i-1}^j\Big], \quad (24)$$

where f_i^j denotes the discretized $f(x_i, t_j)$ and

$$\beta \equiv \alpha^2 \left(\frac{\Delta t}{\Delta x^2}\right). \quad (25)$$

Equation (24) is known as the *explicit Euler* scheme, and can be solved directly for $f(x, t+\Delta t)$ since the right hand side is all known at time step t.

Implicit schemes

As noted in section D, applying the forward difference approximation for the first derivative in time—derived from the Taylor expansion of a function around the point $x+h$—yields the explicit Euler scheme. However, as we will see in section G, the forward method has a rather weak stability criterion which necessitates the use of very small time steps to ensure the solution remains well-behaved.

If we instead employ a backward difference approximation in time (solving Eq. (21) for an $\mathcal{O}(h)$ approximation of the first derivative) we obtain equations sets which are stable for any combination of Δx and Δt. This allows us to use more reasonable time steps and ultimately speeds up and improves the computed solution.

However, using this approximation,

$$f'(t) = \frac{f(t) - f(t-h)}{h} + \mathcal{O}(h), \quad (26)$$

introduces additional complexity in that we are no longer able to simply solve for the unknown $f(x, t+\Delta t)$ in the resulting heat equation approximation. Equating Eqs. (22) and (26) yields the *implicit Euler* scheme:

$$\frac{f_i^j - f_i^{j-1}}{\Delta t} = \alpha^2 \frac{f_{i+1}^j - 2f_i^j + f_{i-1}^j}{\Delta x^2}$$
$$f_i^{j-1} = -\beta\big[f_{i+1}^j + f_{i-1}^j\big] + \big(1+2\beta\big) f_i^j. \quad (27)$$

We note that the only known quantity is the left hand side $f_i^{j-1} = f(x, t - \Delta t)$, which means we must solve all the equations for every spatial point i simultaneously in order to ensure they are all satisfied. This gives rise to a matrix-vector equation in the place of the single algebraic equation needed for the explicit scheme. Denoting $\gamma \equiv 1 + 2\beta$ we may write the equation set corresponding to the integration from $t - \Delta t$ to t as

$$A\mathbf{f}^j = \mathbf{f}^{j-1}, \tag{28}$$

with (zeros ommitted)

$$A = \begin{pmatrix} \gamma & -1 & & & & \\ -1 & \gamma & -1 & & & \\ & -1 & \gamma & -1 & & \\ & & & \ddots & & \\ & & & -1 & \gamma & -1 \\ & & & & -1 & \gamma \end{pmatrix} \tag{29}$$

and

$$\mathbf{f}^j \begin{pmatrix} f_0^j \\ f_1^j \\ f_2^j \\ \vdots \\ \mathbf{f}_N^j \end{pmatrix}, \quad f^{j-1} = \begin{pmatrix} f_0^{j-1} \\ f_1^{j-1} \\ f_2^{j-1} \\ \vdots \\ f_N^{j-1} \end{pmatrix} \tag{30}$$

As the matrix A is tri-diagonal, efficient linear scaling solution algorithms exists which circumvent the traditional solution schemes, e.g. LU decomposition at $\mathcal{O}(N^3)$.

The Crank-Nicolson scheme

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Stability analysis

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Finite element methods

In the case of the finite difference approximation, the derivatives in the original partial differential equation is transformed into a set of algebraic equations by application of *finite difference* approximations arising from Taylor expansions of the solution function. A fundamentally different, and somewhat more sophisticated, solution strategy entails writing the original differential equation in its *weak form*. This is done by integrating the original equation multiplied by a test function $v(x)$ over the domain in question. Such a *finite element* (FEM) approach is normally accompanied by a simple finite difference scheme in time when partial differential equations are considered.

Functional spaces and weak forms

Let us consider the vector space of *all* functions $f : \mathbb{R}^2 \to \mathbb{R}$, which we will call $A(\mathbb{R}^2)$. Obviously, the solution to the heat equation is contained in this space, being a real-valued function of two parameters, x and t. However, we may consider subspaces with more useful structure than the all-encompassing $A(\mathbb{R}^2)$. Assuming the initial and boundary conditions are specified in a sufficiently non-pathological way, the solution at each instant in time will be continuous across the spatial integration domain, Ω. In other words, the solution (at some $\tau \geq 0$) $u(x; \tau) \in C(\Omega)$: The vector space of *continuous* functions $f : \Omega \to \mathbb{R}$. As the heat equation involves a second spatial derivative, we might assume that the solution be twice continuously differentiable, i.e. $u(x; \tau) \in C^2(\Omega)$.

It turns out that demanding $u \in C^2(\Omega)$ is too strict a bound on the behavior of the solution. For example, a valid solution may be

not differentiable on a subset $S \subset \Omega$ with measure zero[1], known as differentiable almost everywhere (a.e.). It is more useful to consider *weakly differentiable* functions, which encompass the ordinary (strongly) differentiable functions and includes other possible solutions of partial differential equations.

Equipping our functional space with the L^1 integral inner product (and the associated norm), consider now a family of functions with compact support[2] $\phi \in C^\infty(\Omega)$, with $\phi(\partial\Omega) = 0$, where we denote by $\partial\Omega$ the boundary of the integration domain (in one dimension, this is just the end two points). Any function $f : \Omega \to \mathbb{R}$ which is bounded on Ω and thus locally integrable (in the Lebesgue sense) on Ω defines a linear functional Λ_f on $C^\infty(\Omega)$ through

$$\Lambda_f(\phi) \equiv \int_\Omega \mathrm{d}x\, f(x)\phi(x). \tag{31}$$

This integral is well defined for any $\phi \in C^\infty(\Omega)$ since $\phi(x)$ is assumed to vanish outside of a compact subset of Ω. Essentially, multiplication with a well-behaved function ϕ and integration over Ω results in something that is more well-behaved than f itself. Next, consider the functional defined by the derivative of f,

$$f'(x) \equiv \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}. \tag{32}$$

Integrating by parts and remembering that $\phi(\partial\Omega) = 0$, we find

$$\begin{aligned}\Lambda_{f'}(\phi) &= \int_\Omega \mathrm{d}x\, f'(x)\phi(x) \\ &= -\int_\Omega \mathrm{d}x\, f(x)\phi'(x).\end{aligned} \tag{33}$$

Note carefully that the first integral is only defined if the limit exists and converges for a.e. $x \in \Omega$, but the last integral is defined and well-behaved for any locally integrable (bounded) function f. This functional exists even if Eq. (32) does not converge point wise for *any* $x \in \Omega$(!).

In the spirit of this, we define the *weak derivative*: The k-th order weak derivative of u is defined as a function v for which the following holds

$$\int_\Omega \mathrm{d}x\, v(x)\phi(x) = (-1)^k \int_\Omega \mathrm{d}x\, u(x)\frac{\partial^k \phi}{\partial x^k}, \tag{34}$$

where a factor (-1) is picked up for each integration by parts.

In defining a weak derivative, we have moved the previous demands on f onto a set of test functions $\phi(x)$. Even if the demands on the functions ϕ are stronger than the previous demands on f ($\phi \in C^\infty(\Omega)$ [infinitely continuously differentiable] as opposed to $f \in C^2(\Omega)$ [twice continuously differentiable]), it will turn out that this is easier to control when considering finite element schemes.

The functional spaces containing functions $f : \mathbb{R}^m \to \mathbb{R}$ with f being k times weakly differentiable are called *Sobolev* spaces, and denoted H^k.

The heat equation in variational (weak) form

As we have seen, the solution function of the original heat equation resides in the Sobolev space $H^2(\Omega)$ of a.e. continuous functions on the domain Ω with weak derivatives up to and including order two. Notice that if the strong (ordinary) derivative exists, then the weak and the strong derivatives are equal. This means that $H^2(\Omega)$ is a *larger* space than $C^2(\Omega)$, which is larger again than the set of infinitely continuously differentiable functions $C^\infty(\Omega)$.

Let us consider a stationary diffusion equation on the form

$$f(x) = \partial_{xx} u(x), \tag{35}$$

with $f(x)$ some known function. Multiplying from the left by a test function $\phi(x) \in C^\infty(\Omega)$ satisfying $\phi(\partial\Omega) = 0$ and integrating over Ω, before integrating by parts, yields a *weak* or *variational* form as

$$\begin{aligned}\int_\Omega \mathrm{d}x\, \phi(x) f(x) &= \int_\Omega \mathrm{d}x\, \phi(x)\frac{\partial^2 u}{\partial x^2} \\ &= -\int_\Omega \mathrm{d}x\, \frac{\partial \phi}{\partial x}\frac{\partial u}{\partial x}\end{aligned} \tag{36}$$

$$\int_\Omega \mathrm{d}x\, \phi(x) f(x) = \int_\Omega \mathrm{d}x\, \frac{\partial^2 \phi}{\partial x^2} u(x). \tag{37}$$

Proving that a solution of the weak equation is also a solution of the strong form is non-trivial and not always possible. We will in the following

[1] A set $S \subset \mathbb{R}$ of measure zeros is a set with vanishing "size," e.g. a finite set of *points*.

[2] The support of a function f is the (closure) of the set of points in the domain of f for which $f(x)$ does *not* vanish, i.e. the set $\{x \in \Omega : f(x) \neq 0\}$.

take if for granted that this is possible and true, or that when it is not possible we will simply interpret the weak solution as *the solution*.

Basis choice and the Galerkin method

So far we have ostensibly only introduced a second unknown function into the problem, without gaining much besides relaxation of some demands on the solution function u. We will now employ the Galerkin method—introducing a basis for $\phi(x)$—and derive a set of algebraic equations which may be solved to obtain an approximation to u.

In order to make the problem numerically tractable, we are now forced to choose a suitable subspace V of $H^2(\Omega)$ to work in. In practice, this is done by introducing a basis $\{\psi_i(x)\}_{i=1}^n$ and letting the subspace be $V_n = \mathrm{Span}(\{\psi_i(x)\})$. In order to simplify the resulting equations, we choose a set of piecewise differentiable functions which vanish exactly on the boundary $\partial\Omega$. One of the simplest choices are the "hat functions,"

$$\psi_i(x) = \begin{cases} 0 & x \in (-\infty, x_i - h) \\ \frac{x-(x_i-h)}{h} & x \in [x_i - h, x_i) \\ 1 - \frac{x-x_i}{h} & x \in [x_i, x_i + h) \\ 0 & x \in [x_i + h, \infty) \end{cases} \tag{38}$$

i.e. piecewise linear functions which are non-zero in the range $[x_i - h, x_i + h]$, increasing linearly to unity from $x_i - h$ to x_i, before decreasing linearly back to zero between x_i and $x_i + h$.

In general, it is numerically beneficient to introduce a spatial discretization—as done in the context of the finite difference method—and choosing a set of functions $\psi_i(x)$ which vanish on all but one of the discretized points. A convenient choice often employed is the Lagrange interpolating polynomials, of which the piecewise first degree functions Eq. (38) are an example. Introducing a discretization, we may group adjacent discretized points into what are known as *elements*. In the case of the Lagrange interpolating polynomial basis, the number of points (or nodes) in each element is determined by the order of the polynomials used. We will in the following refer to the interpolating polynomial bases as Pk-bases, with k denoting the order of the piecewise polynomial functions. Equation (38) represents P1 elements. In general, the Lagrange interpolating polynomials which vanish exactly on all points x_i, except for a single x_k on which it takes the value unity may be written as

$$L_k(x) = \prod_{i \neq k} \frac{x - x_i}{x_k - x_i}. \tag{39}$$

Using Pk elements, each spatial element will contain $k + 1$ nodes, of which two will coincide with nodes in the neighboring elements. Examples of P1, P2, and P3 basis functions are shown in Fig. 1.

Working from the weak form Eq. (36) (we may also choose to work from Eq. (37), but the former yields tidier mathematics), we express the arbitrary $\phi(x)$ in terms of the chosen basis functions $\{\psi_i(x)\}$. It is obvious that *within the subspace* V_n, demanding that Eq. (36) is satisfied *for any* $\phi(x)$ is the same as demanding that it holds for any of the basis functions $\psi_i(x)$. To realize this, let us choose the arbitrary $\phi(x)$ to be $\psi_k(x)$, and express $u(x)$ in terms of its basis expansion in ψ_is:

$$\int_\Omega \mathrm{d}x\, \psi_k(x) f(x) = -\int_\Omega \mathrm{d}x\, \frac{\partial \psi_k}{\partial x} \frac{\partial u}{\partial x}$$
$$\int_\Omega \mathrm{d}x\, \psi_k(x) f(x) = -\int_\Omega \mathrm{d}x\, \sum_i c_i \frac{\partial \psi_k}{\partial x} \frac{\partial \psi_i}{\partial x}.$$

If we do this once for each ψ_k, we end up with a set of n equations for the coefficients c_i, which determine the solution function $u(x)$ *in the subspace* V_n. It is clear that this may be formulated in terms of a matrix-vector equation as $K\mathbf{c} = \mathbf{f}$, with

$$K_{ij} \equiv -\int_\Omega \mathrm{d}x\, \frac{\partial \psi_i}{\partial x} \frac{\partial \psi_j}{\partial x} \tag{40}$$

and

$$\mathbf{f}_i \equiv \int_\Omega \mathrm{d}x\, \psi_i(x) f(x). \tag{41}$$

For smart choices of discretization, the integrals over the basis functions may be easily calculated in closed form.

Pairing the FEM method with a finite difference scheme in time

In order to solve a time dependent problem, a finite difference scheme is traditionally introduced in the temporal dimension. In principle we may choose any such scheme, but we will in the following focus on the simplest such scheme: The forward euler (as derived in section D and Eq. (24)). Modifying the left hand

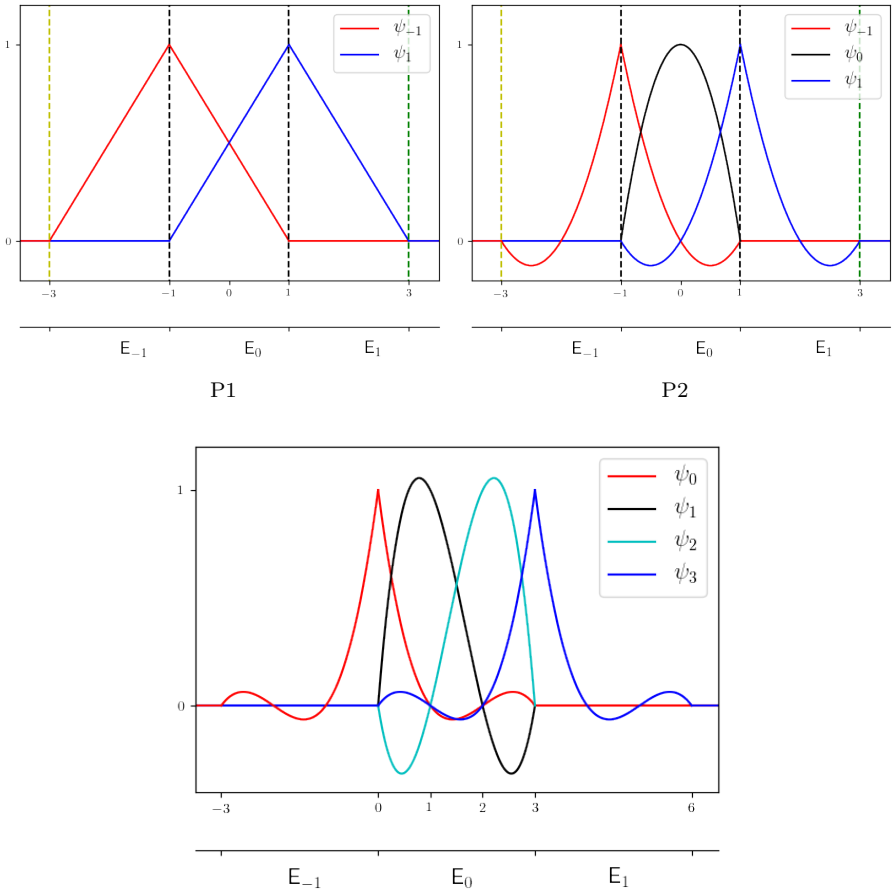

FIG. 1. Piecewise Lagrange interpolating polynomial basis functions for P1, P2, and P3 elements, respectively. The basis functions $\psi_i(x)$ are labelled according to the only discretized spatial point (node) they do not vanish on. Note that all $\psi_i(x)$ take the value $\psi_i(x_i) = 1$ on the one node for which they do not vanish.

side of the stationary diffusion equation—Eq. (35)—and introducing the backwards difference approximation of Eq. (26), we find

$$\begin{aligned}\frac{\partial u}{\partial t} &= \alpha \frac{\partial^2 u}{\partial x^2} \\ \frac{u^{n+1} - u^n}{\Delta t} &= \alpha \frac{\partial^2 u^n}{\partial x^2} \\ u^{n+1} &= u^n + \beta \frac{\partial^2 u^n}{\partial x^2}. \end{aligned} \tag{42}$$

where superscripts denote the discretized time step, i.e. $u(x, t_n) = u^n(x)$. Introducing the matrix

$$M_{ij} \equiv \int_\Omega \mathrm{d}x\, \psi_i(x)\psi_j(x), \tag{44}$$

we may write Eq. (43) as $M\mathbf{c}^{n+1} = M\mathbf{c}^n - \beta K \mathbf{c}^n$. Note carefully that all of the right hand side is known at time step n, which means we may simply directly solve this equation for the coefficients of the solution function relative to the $\{\psi_i\}_i$ basis at time step $n+1$. A similar derivation yields the corresponding equation set for the *backwards* Euler scheme as

$$\Big(M + \beta K\Big)\mathbf{c}^{n+1} = M\mathbf{c}^n. \tag{45}$$

Solving differential equations with neural networks

A part of this project is exploring partial differential problems with the use of deep learning. Deep neural networks (DNN) are networks that consist of two hidden layers or more. In this part of the project we will be using Googles open-source library Tensorflow. A time consuming part of neural networks is the backpropagation and its equations. By using Tensorflow we avoid having to solve the derivatives of a cost function w.r.t to all the different weights and biases. It also provides us with a wast amount of applications. Different activation functions, gradients, loss function etc. And it is easy to add layers and nodes.

In our approach on solving differential equations with DNN we will discover there are some similarities between this new method and the well known finite difference methods. For instance we will not only be using backpropagation as a way of teaching the network, we will also calculate the spatial and temporal derivatives directly, much like we do in the finite difference scheme.

However in the finite difference method we assume the problem can be rewritten into new numerical solvable equations using Taylor expansion, as mentioned above. Tensorflow uses a method called automatic differentiation (AD) to find the derivatives.

AD is a technique for solving derivatives by recognising elementary functions f_i, (like addition, multiplication, exponential, trigonometric functions) which have known derivatives, within more complex functions f.

$$f = f_1 \cdot f_2 \cdot f_3 \cdot f_4 \cdot ... \cdot f_n \tag{46}$$

A composition consisting of a finite set of elementary operations for which the derivatives are known, combining the derivatives of the constituent operations through the chain rule gives the derivative of the overall composition. This is done through a process called forward and reverse mode. Eq. (47) show how we can repeatedly substitute the derivative of the inner functions in the chain rule of a function y w.r.t one fixed variable x. In reverse mode one substitutes the outer functions in the chain rule. A computational graph showing the forward mode of $y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$ is given in Fig. 2.

Calculating the gradients of u with AD gives more accurate results than using Taylor expansion approximations, which is the reason Tensorflow uses AD. In theory, AD should yield exact derivatives to numerical precision (since each step in the chain rule differentiation is exact).

We now use the Galerkin method: Introducing a Pk basis $\{\psi_i(x)\}_i$, left-multiplying by $\psi_k(x)$ and integrating over Ω, performing a single integration by parts on the right hand side:

$$\int_\Omega \mathrm{d}x \sum_i c_i^{n+1}\psi_i(x)\psi_k(x) = \int_\Omega \mathrm{d}x \sum_i c_i^n \psi_i(x)\psi_k(x) - \beta \int_\Omega \mathrm{d}x \sum_i c_i^n \frac{\partial \psi_k}{\partial x}\frac{\partial \psi_i}{\partial x}, \tag{43}$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial \omega_{n-1}}\frac{\partial \omega_{n-1}}{\partial x} = \frac{\partial y}{\partial \omega_{n-1}}\left(\frac{\partial \omega_{n-1}}{\partial \omega_{n-2}}\frac{\partial \omega_{n-2}}{\partial x}\right) = \frac{\partial y}{\partial \omega_{n-1}}\left(\frac{\partial \omega_{n-1}}{\partial \omega_{n-2}}\left(\frac{\partial \omega_{n-2}}{\partial x}\right)\right) = ... \tag{47}$$

We will look at a standard DNN with n hidden layers called multilayer perceptron (MLP). Each layer may contain different number of neurons, or nodes. A complete description of the MLP is given in project 2 and particulary Fig. 1 is explanatory.

Usually in supervised learning we would have one set of training data from where we computed an output and evaluated the cost function. In our case we do not have a correct output value to compare with, we are looking at a partial differential equation and our data set is simply initial values of x and t which we try to fit to the model. But from Eq. (48) we see that we seek a solution $u(x,t)$ that minimizes the right hand side of the equation.

$$0 = \frac{\partial u(x,t)}{\partial t} - \frac{\partial^2 u(x,t)}{\partial x^2} \tag{48}$$

It therefore makes sense to select the mean squared error (MSE) as our cost function as it allow us to do exactly so. Usually one has a set of test data to check how well the network is able to predict the outcome. However we have the exact analytical solution to check with directly.

$$MSE = \frac{1}{n}\sum^{n}\left(\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2}\right)^2 \tag{49}$$

Unlike finite element where we have algebraic equation sets to solve iterally, with DNN we must introduce a trial function, a guess on the solution, in order to give the network something to work on. Our next task is therefore designing a trial function that can be used as input for the cost function. Preferably a trial function that satisfy the initial conditions and boundary conditions and isn't to complicated. If we look at Eq. (50) we see that for $x = 0$ and $x = 1$ the trial function is zero since $\sin(0) = \sin(\pi) = 0$ which is the boundary conditions. For $t = 0$ it is $\sin(\pi x)$ which is the initial condition. And Eq. (50) fulfilles all conditions.

$$u_{trial}(x,t) = (1-t)\sin(\pi x) + x(1-x)tN(x,t,P) \tag{50}$$

The trial function is there to assure the network finds a solution that fulfilles the conditions we are given. Otherwise it could just find any function $u(x,t)$ where its first time derivative matched its second spatial derivative. $N(x,t,P)$ is a function that describes the output of the network given the x, t and P, the parameters of the network (weights and biases).

Now all we need is an optimizer in order to asjust the parameters. Tensorflow has several possible optimizer methods. However we will look at ordinary gradient descent.

RESULTS AND DISCUSSION

We now turn to the last part of this project and try to solve the diffursion equation with a neural network. We use Tensorflow to solve this problem and the structure of the network can be chenged by adding new layers in the python list called num_hidden_neurons, where each element is the number of nodes in each layer. Our network takes as input 100 pairs of x and t-values. These are then updated and optimized with gradient descent. First off the Tensorflow optimizer AdamOptimizer was tested. This method was used in project 2 giving good results. Observing an increase in the error compared to GradientDescentOptimizer however, the method was droped in this case, and gradient descent has been used through out this section. After running the network with a number of different structures we observe that adding layers may not necessarily give better results. If we look at Table I, just increasing the amount of nodes in a NN with one layer essentially becomes more computationally expensive. The more nodes and layers we add the more time consuming the network becomes. For 90 nodes we reach an optimal point with a maximum difference between analytical solution and the final output, ϵ_{diff}, of the NN of 0.0179.

$$\epsilon_{diff} = u_{analytic}(x,t) - u_{NN}(x,t,P) \tag{51}$$

We try to add a new layer also containing 90 nodes. This give a better result, see Table III. Compared to deeper networks with fewer nodes it is also better, but takes longer time than just increasing the number of iterations. Running one hidden layer containing 90 node with 200.000 iterations give a decrease in error and computational time compared to two layers with 90 nodes, see Table II. It looks as though larger single layers is more important than increasing the number of layers and creating a deeper network. This is consistent with the universal approximation theorem.

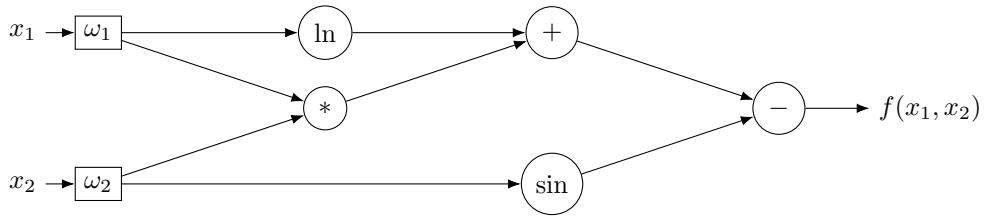

FIG. 2. A representation of the forward accumulation of $f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$. We can give $\ln(\omega_1)$ a new label, ω_3, and the multiplication $\omega_1\omega_2$ we can call ω_4 and so forth. Relabeling each new elementary operation and taking its derivative is the forward mode with the final varible becoming $\omega_5 = \omega_4 - \omega_3$ and $\dot{\omega_5} = \dot{y}$, the derivative of the original complex function.

TABLE I. Simulation of NN with one hidden layer and number of neurons given in table. Number of iterations equal 100.000 and learning rate at 0.01. Results given as average over 5 runs. It is the difference between the analytical solution and the final output of the network, its maximum value and the mean, that is listed.

Number of neurons	max $\lvert\epsilon_{diff}\rvert$	mean $\lvert\epsilon_{diff}\rvert$	Computational time [s]
10	0.0263	0.00097	24.0
40	0.0420	0.00012	27.7
80	0.0186	0.00074	33.2
90	0.0179	0.00056	35.4
100	0.0187	0.00077	36.8

TABLE II. Simulation of NN with one hidden layer and 90 neurons. Number of iterations equal 200.000 and learning rate at 0.01. Results given as average over 3 runs.

Number of neurons	max $\lvert\epsilon_{diff}\rvert$	mean $\lvert\epsilon_{diff}\rvert$	Computational time [s]
90	0.0113	0.00036	1m 9s

TABLE III. Simulation of NN with the number of hidden layer and neurons given in table, [10,10] equals two layers with 10 nodes in each. Number of iterations equal 100.000 and learning rate at 0.01. Results given as average over 3 runs.

[layer$_n$,]	max $\lvert\epsilon_{diff}\rvert$	mean $\lvert\epsilon_{diff}\rvert$	Computational time [s]
[10,10]	0,0844	0.03736	33.1
[10,10,10]	0,1334	0,02409	42.0
[10,10,10,10]	0.0156	0.00111	50.2
[90,90]	0.0134	0.00098	1m 31s

Comparing NN with FD and FEM

We want to compare the solution using the neural network with the traditional PDE-methods

of finite difference and finite element. We have implemented three finite-difference solutions, namely the explicit and implicit Euler as well as the Crank-Nicholson scheme. Furthermore we implemented a FEM P2 solution.

In order to compare the solutions we solved the one-dimensional diffusion equation described in the theory section with closed form soultion given by Eq. (17). The solution was computed in the time interval $t \in [0, 0.1]$, where the solution has diffused but is not zero. We compare the different solutions at the end of the simulation, namely at $t = 0.1$.

We used a rather coarse spatial grid with only $N_x = 10$ integration points in space and $N_t = 25$ integration points in time. The parameters were chosen such that the explicit Euler scheme is stable and we could compare all methods, with the same parameters. For the neural network a single layer with $N_{neurons} = 90$ hidden neurons. The network was trained for $N_{iterations} = 10^5$ iterations with the trial function given by Eq. (50) and using the method of gradient descent for optimization. The training phase completed in roughly 4 minutes.

Fig. (3) shows the computed solution at $t = 0.1$ using all methods compared with the closed form solution. Furthermore, we have plotted the deviation from the exact solution for all methods, as it is to some degree easier to see difference between the methods in this way.

It is clear that with the chosen parameters the neural network and the Crank-Nicholson performs best in terms of accuracy, with a slight better performance using the NN. However, it should be noted that the Crank-Nicholson scheme takes merely a second to complete in contrast with the 4 minutes used to train the network. Thus, in terms of accuracy pr. wall time, the Crank-Nicholson scheme is superior.

Nevertheless, this exercise provides a proof-of-principle that we can use neural networks to solve a partial differential equation. However, it seems far away that it can outperform the standard approaches in terms of efficiency, even though it provides adequate accuracy.

Increasing the number of time steps

Keeping our attention at the temporal region $[0, 0.1]$, we now explore how the network solution compares to the traditional methods when the number of time steps is increased. We run simulations using the P2 finite element scheme and train the network using $N_t \in [10, 1000]$. As the training is rather slow, we are not able to explore fully the upper regions of this interval in the latter case. In contrast, the FEM method easily calculates the solution at $t = 0.1$ in less than a second, even when using $N_t = 1000$. Note that this corresponds to a time step of $\Delta t = 0.0001$. Plotting the mean difference between the numerical solution and the closed form solution yields the plot shown in Fig. 4.

CONCLUSION

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

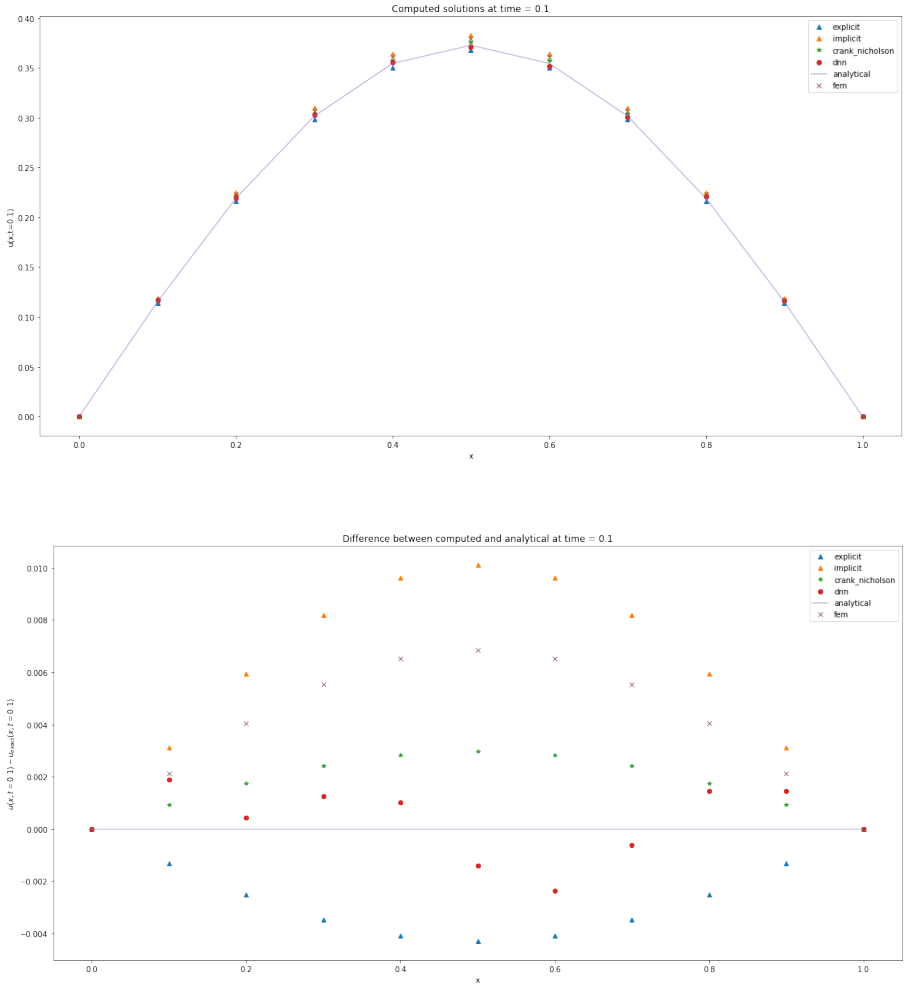

FIG. 3. Comparison of the solution of the diffusion equation using a neural network, finite difference schemes and the finite element method with P2 functions.

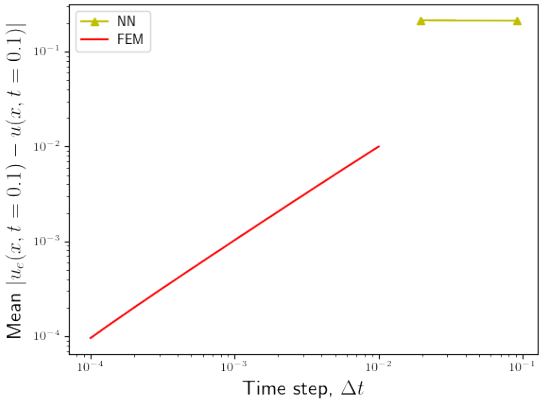

FIG. 4. Comparison of the error scaling between the P2-FEM scheme and the NN method, for time steps in the range $10^{-4} \leq \Delta t \leq 10^{-1}$. We note that the traditional scheme improves steadily as Δt decreases, while this is not seen for the NN solution.