A comparison between neural networks and the finite element method to solve partial differential equations
FYS-STK4155 - Project 3

Lars Bosch, Philipp Brückelt, Thomas Engl*
University of Oslo
(Dated: December 15, 2025)

Partial differential equations (PDEs) have a lot of applications in mathematics and natural sciences, but can usually not be solved analytically. Therefore, we analyze machine learning methods to solve partial differential equations numerically. To this end, we construct a physics-informed neural network (PINN), i.e., a neural network taking the physics of the underlying PDE into account by imposing a trial solution and boundary conditions as part of the network architecture. We use this to solve the heat equation in one and two dimensions. In addition, we also solve this problem using the finite element method (FEM) and compare the results afterwards. Our experiments show that our PINN is compatible with FEM. Depending on parameters chosen for both methods, the PINN can even achieve lower L^2 and L^∞ errors. However, the computation time is significantly higher for the PINN than for FEM.

I. INTRODUCTION

Partial differential equations (PDEs) [1–4] are a powerful mathematical tool to describe physical processes and are frequently used to model, e.g., diffusion, fluid dynamics or quantum mechanics. The study of PDEs is one of the largest research areas in Analysis, not only due to their importance but also because solving PDEs is in general, a highly complicated or even intractable task. While for many PDEs we can prove well-posedness, i.e., existence, uniqueness and continuous dependence on the data, often analytical solutions do not exist. However, for many PDEs not even the former is known. For example, it is still one of the biggest unsolved questions, one of the Millenium problems, whether the Navier-Stokes equations [5] which describe the motion of viscous fluids, admit a global smooth solution in three dimensions.

In the case where existence and uniqueness is known, but analytic solutions are hard or impossible to find, several methods have been developed in the last century to compute numerical PDE solutions, with the Finite Element Method (FEM) [3, 6, 7] being one of the most powerful.

Over the last years, machine learning methods have gained more and more importance in many research areas, also in Numerical Analysis. In [8] and [9] Raissi, Perdikaris and Karniadakis introduce physics informed neural networks and demonstrate how to compute data-driven solutions to partial differential equations. Originally, using neural networks to solve PDEs was motivated by the fact, that numerical methods suffer from the curse of dimensionality while the approximation error of neural networks does not increase drastically with the dimensionality. However, in this work we restrict to low-dimensional problems. In fact, we study the heat equation

$$\partial_t u - \kappa \Delta u = f$$

with $\kappa > 0$ in one and two dimensions. We explain now the architecture of a PINN for the one dimensional heat equation with $f = 0$, boundary conditions $u(0,t) = 0$ for $t \geq 0$ and $u(1,t) = 0$ for $t \geq 0$. Given an initial condition $u_0(x) = u(x,0)$ for $0 < x < 1$ we can define a trial solution

$$e^{-t} \cdot u_0(x) + x \cdot (1-x) \cdot t \cdot N(x,t)$$

with N a neural network w.r.t t and x. The trial solution ensures that initial and boundary conditions are satisfied. The neural network can now be learned to minimize the residuals of the PDE which corresponds to the loss

$$\mathcal{L}_{\text{PDE}} = \frac{1}{n} \sum_{i=1}^{n} \left(\partial_t u(x_i,t_i) \; - \; \kappa \, \partial_{xx} u(x_i,t_i)\right)^2 .$$

Our goal is to find a neural network architecture s.t. the approximated solution of the PDE given by the PINN comes close to the analytic and the FEM solution.

There are also other ways to solve PDEs using machine learning methods. Some of them are presented in [10].

The remainder of this work is structured as follows. In Section II we describe the initial boundary value problem to be studied. We also present analytic solutions to concrete examples and derive the weak formulation. Afterwards, we give an overview of physics informed neural networks in Section III. In Section IV we discuss the implementation of FEM, the neural network in Python and L-BFGS. The results of our numerical experiments are presented in Section V. Finally, we summarize our main findings in Section VI.

* https://github.com/thomas-engl/Project-3-Applied-ML

II. PDE

A. Problem setting

We study the heat equation (see e.g., [11], [1, Sc. 2.3] or [2, Sc. 10.1]) in one and two spatial dimensions and consider the initial boundary value problem (IBVP)

$$\begin{cases} \partial_t u - \kappa\Delta u &= f \quad \text{in } \Omega \times (0,T], \\ u &= 0 \quad \text{on } \partial\Omega \times (0,T], \\ u(0,\cdot) &= u_0 \end{cases} \tag{1}$$

where Ω is open and bounded, $f \in L^2(\Omega \times (0,T])$, $u_0 \in H^1(\Omega)$ and $\kappa > 0$ is the thermal diffusivity. Here, the Laplacian Δ is taken with respect to the spatial coordinates. $T < \infty$ is a finite time horizon and we mainly work with $T = 0.2$.

In our numerical experiments we work with the concrete IBVP (1) with $\Omega := (0,1)^2$ and the initial function u_0 given by

$$\begin{aligned} u_0(\mathbf{x}) &= u_0(x,y) \\ &= \sin(\pi x)\sin(\pi y) + \sin(2\pi x)\sin(4\pi y) \\ &=: u_0^1(x,y) + u_0^2(x,y) \end{aligned}$$

in the two-dimensional case. For the 1D example, we consider $\Omega = (0,1)$ the unit interval and

$$u_0(x) = \sin(\pi x) + \sin(4\pi x).$$

We use $f \equiv 0$ or $f(x) = \sin(\pi x)$ as right hand side. Note that f is not time-dependent. Under these conditions the IBVP (1) admits a unique smooth solution [2, Sc. 10.1].

B. Analytic solution

We only compute the analytic solution in the 2D case. In one dimension, it is then straight-forward to find the analytic solution by the same approach. To find a solution to (1), we first solve the homogeneous heat equation, i.e., with $f \equiv 0$. Here, we follow [1, Sc. 4.1] and use separation of variables. To this end, we consider two IBVP with initial conditions u_0^1 and u_0^2 separately. This is reasonable because in general the solution to the homogeneous heat equation is given by a linear combination of eigenfunctions. We use the ansatz

$$u(t,\mathbf{x}) = v(t)w(\mathbf{x})$$

for $t \in [0,T], \mathbf{x} \in \Omega$. The dynamics in (1) is satisfied if and only if

$$\frac{v'(t)}{v(t)} = \mu\frac{\Delta w(\mathbf{x})}{w(\mathbf{x})}$$

for some $\mu \in \mathbb{R}$.

It is easy to see that

$$\begin{aligned} -\Delta u_0^1(x,y) &= -\partial_{xx}^2 u_0(x,y) - \partial_{yy}^2 u_0(x,y) \\ &= 2\pi^2 \sin(\pi x)\sin(\pi y) \\ &= 2\pi^2 u_0^1(x,y), \end{aligned}$$

and

$$-\Delta u_0^2(x,y) = 20u_0^2(x,y),$$

i.e., u_0^1 and u_0^2 are eigenfunctions of the elliptic operator $-\Delta$, with homogeneous Dirichlet boundary conditions, to the eigenvalues $2\pi^2$ and $20\pi^2$, respectively. Thus, $w = u_0$ and $\mu = -2\pi^2$ for u_0^1 which yields

$$v(t) = c\mathrm{e}^{-2\kappa\pi^2 t}$$

for some $c \in \mathbb{R}$. We have the analogous result for the IBVP with initial condition u_0^2. Considering the initial condition, we find that $c = 1$ and

$$u^1(t,\mathbf{x}) := \mathrm{e}^{-2\kappa\pi^2 t} u_0^1(\mathbf{x})$$

solves (1) for u_0^1. The solution with initial condition u_0^2 is found similarly,

$$u^2(t,\mathbf{x}) := \mathrm{e}^{-2\kappa\pi^2 t} u_0^2(\mathbf{x}).$$

Hence, the solution for the actual IBVP with $f \equiv 0$ is

$$\begin{aligned} u(t,\mathbf{x}) &= u^1(t,\mathbf{x}) + u^2(t,\mathbf{x}) \\ &= \mathrm{e}^{-2\kappa\pi^2 t}\sin(\pi x)\sin(\pi y) \\ &\quad + \mathrm{e}^{-20\kappa\pi^2 t}\sin(2\pi x)\sin(4\pi y). \end{aligned}$$

Now, we proceed with the right hand side $f(x,y) = \sin(\pi x)\sin(\pi y)$. An approach to find a solution is shown in [4, Sc. 7.2]. Following this, we compute an *equilibrium solution* u_E, i.e., a solution to the Poisson equation

$$-\kappa\Delta u = f$$

with 0 boundary conditions which is easy by the choice of f. The solution is given by

$$u_E(x,y) = \frac{1}{2\pi^2\kappa} f(x,y).$$

Next, we consider

$$v(t,x,y) := u(t,x,y) - u_E(x,y),$$

that is,

$$\begin{aligned} v(0,x,y) &= u_0(x,y) - u_E(x,y) \\ &= \left(1 - \frac{1}{2\kappa\pi^2}\right)\sin(\pi x)\sin(\pi y) \\ &\quad + \sin(2\pi x)\sin(4\pi y) \end{aligned} \tag{2}$$

and solve (1) for v with $f \equiv 0$ and initial condition given by (2). We have done this before and know that

$$v(t,x,y) = \left(1 - \frac{1}{2\kappa\pi^2}\right)\sin(\pi x)\sin(\pi y)\mathrm{e}^{-2\kappa\pi^2 t} + \sin(2\pi x)\sin(4\pi y)\mathrm{e}^{-20\kappa\pi^2 t}. \tag{3}$$

Hence, the solution to the original IBVP (1) is given by

$$\begin{aligned} u(t,x,y) &= v(t,x,y) + u_E(x,y) \\ &= \left(1 - \frac{1}{2\kappa\pi^2}\right)\sin(\pi x)\sin(\pi y)\mathrm{e}^{-2\kappa\pi^2 t} \\ &\quad + \sin(2\pi x)\sin(4\pi y)\mathrm{e}^{-20\kappa\pi^2 t} \\ &\quad + \frac{1}{2\kappa\pi^2}\sin(\pi x)\sin(\pi y). \end{aligned} \tag{4}$$

Uniqueness of the solution can easily be shown by maximum principles [1, Sc. 2.3.3.].

As mentioned before, the analytic solution for the 1D example can be computed by the same approach. Note that $\sin(\pi x)$ is an eigenfunction of $-\Delta$ in one dimension with homogeneous Dirichlet boundary conditions on $(0,1)$ with the eigenvalue π^2. Thus,

$$v(t,x) = \mathrm{e}^{-\kappa\pi^2 t}\sin(\pi x) + \mathrm{e}^{-16\kappa\pi^2 t}\sin(4\pi x) \tag{5}$$

solves (1) with $f \equiv 0$. To obtain the solution with $f(x) = \sin(\pi x)$ we compute an equilibrium solution as before and combine the solutions to find that

$$u(t,x) = \left(1 - \frac{1}{\kappa\pi^2}\right)\sin(\pi x)\mathrm{e}^{-\kappa\pi^2 t} + \sin(4\pi x)\mathrm{e}^{-16\kappa\pi^2 t} + \frac{1}{\kappa\pi^2}\sin(\pi x). \tag{6}$$

Remark 1. *From equations* (3), (4), (5) *and* (6) *we see that solutions of the heat equation decay exponentially in time and become almost stationary for t large enough. The larger κ, the faster the decay is.*

C. Weak formulation and the finite element method

The finite element method is a variational method to solve, in particular, elliptic partial differential equations [7, Ch. 1]. Note that the heat equation is a parabolic PDE but it can be turned into an elliptic problem by discretizing the time separately, as we see in the next steps. Indeed, the heat equation is of the form

$$\partial_t u + Lu = f$$

with L being an elliptic operator.

To solve an elliptic PDE using the finite element method, the equation is transformed into its weak formulation. Its solution lies in a suitable Sobolev space V. But since this space is infinite-dimensional, for numerical computations one restricts to a finite-dimensional subspace V_h, the Finite Element space [6, Ch. 3], leading to a system of linear equations. This is known as the Ritz-Galerkin method [7, Ch. 4]. The solution obtained by FEM interpolates the solution in V by a polynom of chosen degree. It can be shown that the interpolation error vanishes as the mesh size (maximal diameter of the elements) converges to 0 (see e.g. [6, Theorem 4.4.20]).

In the following, we denote by $H_0^1(\Omega)$ the Sobolev space

$$H_0^1(\Omega) := \{u \in L^2(\Omega) \mid \nabla u \in L^2(\Omega) \text{ and } u = 0 \text{ on } \partial\Omega\}$$

with norm

$$\|u\|_{H^1}^2 := \|u\|_{L^2}^2 + \|\nabla u\|_{L^2}^2 = \int_\Omega |u|^2 \,\mathrm{d}\mathbf{x} + \int_\Omega |\nabla u|^2 \,\mathrm{d}\mathbf{x}$$

where $|\cdot|$ denotes the Euclidean norm. The derivatives are understood in the distributional sense. Note that $H_0^1(\Omega)$ is a Hilbert space.

Now, we derive the weak formulation (cf. [12, Sc. 1.3.1]) of the IBVP in arbitrary dimensions. To this end, we first discretize the time derivative. Let $0 = t_0 < t_1 < \cdots < t_n = T$ for some $n \in \mathbb{N}$, where $t_i = i\tau$, $i = 1, ..., n$ and $\tau = \frac{T}{n}$. Denote $u(t_i, \cdot) = u^{(i)}$. Then, we can approximate the time derivative at t_i by the backward difference

$$\partial_t u^{(i)} \approx \frac{u^{(i+1)} - u^{(i)}}{\tau}.$$

Inserted in the dynamics in (1), we obtain

$$\frac{u^{(i+1)} - u^{(i)}}{\tau} - \kappa\Delta u^{(i+1)} = f^{(i+1)}$$

which is equivalent to

$$u^{(i+1)} = \tau\left(f^{(i+1)} + \kappa\Delta u^{(i+1)}\right) + u^{(i)}. \tag{7}$$

This is now an elliptic PDE. Since we know $u^{(0)} = u_0$, we can solve this iteratively. Now, denote $u := u^{(i+1)}$ and $f := f^{(i+1)}$ for simplicity. To find the weak formulation of (7), we multiply by a test function $v \in H_0^1(\Omega)$ and integrate by parts. This yields

$$\begin{aligned} \int_\Omega uv \,\mathrm{d}\mathbf{x} &= \int_\Omega u^{(i)}v + \tau\left(f + \kappa\Delta u\right)v \,\mathrm{d}\mathbf{x} \\ &= \int_\Omega \left(\tau f + u^{(i)}\right)v \,\mathrm{d}\mathbf{x} + \int_{\partial\Omega} \tau\kappa v \nabla u \cdot \vec{n} \,\mathrm{d}S \\ &\quad - \int_\Omega \tau\kappa\nabla u \cdot \nabla v \,\mathrm{d}\mathbf{x} \\ &= \int_\Omega \left(\tau f + u^{(i)}\right)v \,\mathrm{d}\mathbf{x} - \int_\Omega \tau\kappa\nabla u \cdot \nabla v \,\mathrm{d}\mathbf{x}, \end{aligned}$$

where the boundary integral vanishes due to the boundary conditions. Here, $\cdot$ denotes the standard inner product in $\mathbb{R}^d$ and $\vec{n}$ is the outer normal vector. Hence,

$$\int_\Omega uv + \tau\kappa\nabla u \cdot \nabla v \,\mathrm{d}\mathbf{x} = \int_\Omega \left(\tau f + u^{(i)}\right)v \,\mathrm{d}\mathbf{x}.$$

Defining

$$a\colon H_0^1(\Omega)\times H_0^1(\Omega)\to\mathbb{R},$$
$$(u,v)\mapsto\int_\Omega uv+\tau\kappa\nabla u\cdot\nabla v\ \mathrm{d}\mathbf{x}$$

and

$$\ell\colon H_0^1(\Omega)\to\mathbb{R}$$
$$v\mapsto\langle\tau f+u^{(i)},v\rangle_{L^2}:=\int_\Omega\left(\tau f+u^{(i)}\right)v\ \mathrm{d}\mathbf{x},$$

the variational problem to solve becomes

$$a(u,v)=\ell(v)\quad\forall v\in H_0^1(\Omega).\tag{8}$$

Theorem 1. *The variational problem* (8) *admits a unique solution* $u\in H_0^1(\Omega)$.

For a proof in the one-dimensional case, see Appendix A.

III. PHYSICS INFORMED NEURAL NETWORKS

Physics-informed neural networks (PINNs), introduced in [8, 9], can be used to solve partial differential equations of the form

$$\partial_t u+\mathcal{N}[u]=0\tag{9}$$

for $x\in\Omega, t\geq 0$ where $\mathcal{N}$ is a possibly nonlinear differential operator. It is computed using automatic differentiation [13]. In the case of the heat equation, we have $\mathcal{N}[u]=-\kappa\Delta u-f$.

An advantage of PINNs is that they can be used for very different sorts of PDEs. For example, in [8] the authors apply PINNs to solve the Burgers, Schrödinger and Allen-Cahn equations.

In contrast to common neural networks, PINNs are not only driven by data but take the physics of the underlying PDE into account. The goal is to minimize the PDE residual given by

$$R(\hat{u})=\partial_t\hat{u}+\mathcal{N}[\hat{u}]$$

where $\hat{u}$ is an approximation to the PDE solution obtained by the neural network, exploiting the property of neural networks being universal approximators [14]. Based on empirical experience one can expect that this approach leads to sufficiently exact solution as soon as the PDE problem is well-posed [8].

Ideally, $R(\hat{u})=0$ since in this case, $\hat{u}$ satisfies (9). We determine $\hat{u}$ using a trial solution

$$\hat{u}(t,\mathbf{x})=g_1(t)u_0(\mathbf{x})+g_2(\mathbf{x})tN(t,\mathbf{x})$$

where $g_1\colon\mathbb{R}\to\mathbb{R}$ satisfies $g_1(0)=1$, $h\colon\mathbb{R}^d\to\mathbb{R}$ is a function with $g_2(\mathbf{x})=0$ if and only if $\mathbf{x}\in\partial\Omega$ and N is a neural network. This ensures that $\hat{u}$ satisfies the initial and boundary conditions in (1). The function g_1 should be chosen in a suitable way to achieve a good convergence. We took $g_1(t)=\exp(-t)$ it is known that solutions of the heat equation decrease exponentially.

We distinguish two different approaches to construct a PINN, the discrete- and continuous-time approach [8, 10]. In a discrete-time model, the PDE is first discretized in time as we did in Section II C for the implementation in FEM, and the resulting time-independent PDEs are solved by a neural network. In contrast, a PINN following the continuous-time model solves the time-dependent equation in one step. In this work, we always use the latter model.

IV. IMPLEMENTATION

A. FEM solution

To solve the IBVP numerically, we use the finite element method and the software package FEniCSx [15–18] with DOLFINx version 0.9.0. We used simple Lagrange elements [6, Sc. 3.2], triangular elements in the 2D case, of degree 1. The mesh is chosen to be equidistant. To discretize the time derivatives, we chose a step size of $\tau=0.001$ in two dimensions and $\tau=0.002$ for the 1D problem. This is reasonable since it guaranties stability and especially for small times $t<<1$ the solution changes fast.

Moreover, we used the conjugated gradient (CG) method [19] and Algebraic Multigrid (AMG) as preconditioner to solve the linear problem obtained from discretizing (8). This is justified by the following theoretical result, proved in Appendix A.

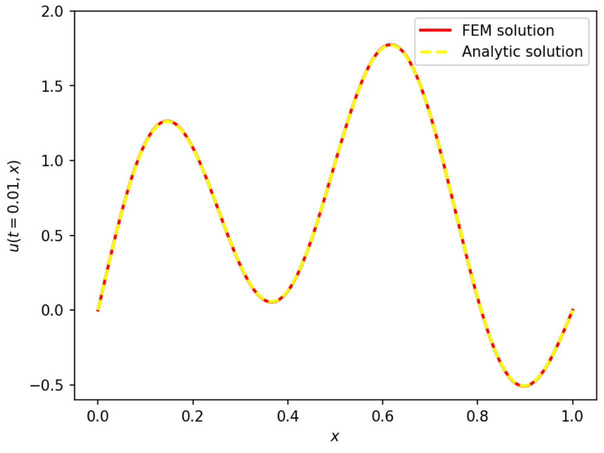
Proposition 1. *The system matrix in the linear problem resulting from* (8) *is positive definite. In particular, the CG method is applicable as solver.*

No parameter tuning was performed since the focus of this work lies on the implementation of the neural network. We only tried to achieve a satisfactory reference solution using FEM.

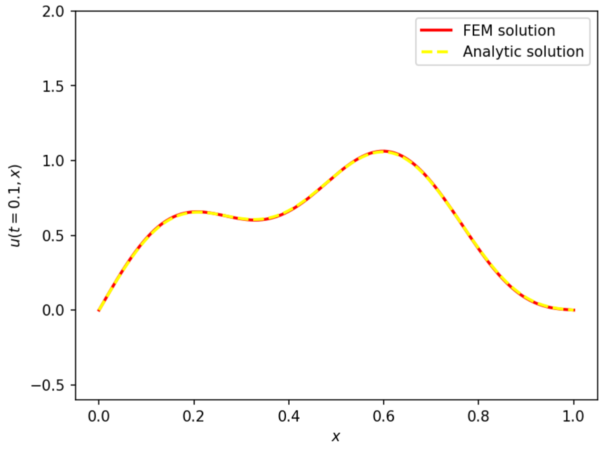
We also created animations of the FEM solution using Pyvista [20] for the 2D animation and matplotlib in one dimension. GIF and MP4 files of the animations can be found on Github[21].

B. Neural network architecture

For the implementation of the PINN, we used pytorch [22]. The training of the network was done by performing a certain number of epochs with the Adam optimizer to reduce the loss roughly, followed by some more epochs using L-BFGS [23], a quasi-Newton method, as it is, e.g., suggested in [24]. The benefits of this approach are



(a) At time $t = 0.01$



(b) At time $t = 0.1$

Figure 1. 1D FEM solution with right-hand side $f(x) = \sin(\pi x)$

demonstrated in Section V. In the subsequent subsection, we explain L-BFGS in more detail.

In one dimension, we worked with 100 grid points in x- and t-direction, resulting in a total of $10,000$ grid points for the training of the neural network. In two dimension, we had to reduce the number of grid points in each dimension to 30 due to memory limitations. This still yields $30^3 = 27,000$ collocation points in total.

We scaled our spatial data points $\mathbf{x}$ from the unit interval $(0,1)^d$ with $d = 1, 2$, to the symmetric interval $(-1,1)^d$ by the affine-linear transformation $\mathbf{x} \mapsto 2\mathbf{x} - 1$.

Moreover, in order to scale the time data, we introduced a learnable scaling,

$$\hat{t} = \alpha t, \quad \alpha = \mathrm{e}^{\theta}$$

with θ being another parameter to train. Hence, the network can itself learn a time scaling fitting to the PDE solution. We did not proceed like this in every experiment, but added the learnable time scale as an optional parameter in our network architecture.

C. L-BFGS

As mentioned before, we will use Limited-Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) [23, 25] as optimizer to train our PINN. Before explaining this in detail, we first recall the update step of Newton's Method for Optimization.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - H_f(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$$

where:

- $\mathbf{x}_k$ is the current iterate (vector of parameters),
- $\nabla f(\mathbf{x}_k)$ is the gradient of f at $\mathbf{x}_k$,
- $H_f(\mathbf{x}_k)$ is the Hessian matrix (second derivatives) of f at $\mathbf{x}_k$.

The explicit computation of the Hessian is very expense and quasi-Newton methods like L-BFGS replace the inverse Hessian by an approximation H_k which is calculated iteratively. Using step length α_k we obtain the update step

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k H_k \nabla f_k.$$

In many cases no line search is needed and $\alpha_k = 1$ leads to good results. For the BFGS algorithm H_k is given by

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T$$

with

$$\rho_k = \frac{1}{y_k^T s_k}, \qquad V_k = I - \rho_k y_k s_k^T,$$

and

$$s_k = \mathbf{x}_{k+1} - \mathbf{x}_k, \qquad y_k = \nabla f_{k+1} - \nabla f_k.$$

The motivation for the choice of this H_k is explained in [25]. Ordinary BFGS calculates in every update step s_k and y_k, uses these values to compute H_{k+1} and stores H_{k+1}. This storage can be problematic for large matrices and motivates limited-memory BFGS. A small proof by induction shows

$$\begin{aligned} H_k = {} & \left(\prod_{j=k-1}^{k-m} V_j^T\right) H_k^0 \left(\prod_{j=k-m}^{k-1} V_j\right) \\ & + \sum_{i=0}^{m-1} \rho_{k-m+i} \left(\prod_{j=k-1}^{k-m+i+1} V_j^T\right) s_{k-m+i} \\ & \cdot s_{k-m+i}^T \left(\prod_{j=k-m+i+1}^{k-1} V_j\right). \end{aligned}$$

For our update step we do not need explicitly H_k but $H_k \nabla f_k$. The following algorithm exploits the previous formula to compute this term in an efficient way.

Algorithm 1 L-BFGS

```
Require: Current gradient $\nabla f_k$, correction pairs $\{(s_i, y_i)\}$ for $i = k-m, \dots, k-1$, scalars $\rho_i = 1/(y_i^T s_i)$, initial scaling matrix $H_k^0$
 1: $q \leftarrow \nabla f_k$
 2: for $i = k-1, k-2, \dots, k-m$ do
 3:     $\alpha_i \leftarrow \rho_i \, s_i^T q$
 4:     $q \leftarrow q - \alpha_i \, y_i$
 5: end for
 6: $r \leftarrow H_k^0 q$
 7: for $i = k-m, k-m+1, \dots, k-1$ do
 8:     $\beta \leftarrow \rho_i \, y_i^T r$
 9:     $r \leftarrow r + s_i(\alpha_i - \beta)$
10: end for
11: return $r$                    ▷ This equals $H_k \nabla f_k$
```

With this algorithm we only have to store the last m correction pairs $\{(s_i, y_i)\}$ and not the whole matrix H_k.

D. Use of LLMs

We mainly used ChatGPT for questions about Python, in particular Dolfinx syntax since the latter has changed a lot in latest versions. Also, we sometimes needed ChatGPT to get along with this LaTex template. The chats can be found on GitHub.

V. RESULTS

A. FEM results

We start the discussion of the results with the finite element method and use the results that we obtain here to compare our neural network with common numerical methods. In the experiments we set $\kappa = 0.1$.

Let us first consider the one-dimensional case. Here, the FEM grid consisted of 128 equally sized sub-intervals of $(0, 1)$ and the time step size was set to $h = 0.002$. We worked on the time interval $[0, T]$ where $T = 0.2$, resulting in a number of 100 time steps. As mentioned in Section II, we worked with two different right hand sides of (1), namely

$$f_1(x) = 0, \tag{10}$$

$$f_2(x) = \sin(\pi x). \tag{11}$$

The numerical solution at time $t = 0.01$ and $t = 0.1$ and f_2 as right hand side is visualized in Figure 1. Table A.1 shows the error achieved by FEM, measured in the L^2 and L^∞ norm, where

$$\|v\|_{L^2(\Omega \times [0,T])} := \left(\int_0^T \int_\Omega |v(t, \mathbf{x})|^2 \mathrm{d}\mathbf{x} \; \mathrm{d}t \right)^{1/2} \tag{12}$$

right hand side	L^2 error	L^∞ error	run time [s]
f_1	0.001218	0.005495	6.169
f_2	0.001218	0.005447	12.581

Table A.1. Results for FEM in 1D

right hand side	L^2 error	L^∞ error	run time [s]
f_3	0.000438	0.003166	20.216
f_4	0.055476	0.276321	13.208

Table A.2. Results for FEM in 2D

and

$$\|v\|_{L^\infty(\Omega \times [0,T])} := \operatorname{esssup}_{(\mathbf{x},t) \in \Omega \times (0,T)} |v(t, \mathbf{x})|. \tag{13}$$

We computed the error as follows. First, the analytic solution is interpolated on the FEM mesh, but the degree of the finite elements is set higher, since the analytic solution is not piecewise linear as the FEM approximation. We chose a degree of 5. Afterwards, the spatial integral in (12) is computed using dolfinx. The integral in time is then obtained by a simple Monte Carlo approximation. To approximate the L^∞ norm, we simply compute (13) on the given mesh.

The errors for different f almost do not differ. This is because we only compute the solutions on a short time interval, where the analytic solutions to the IBVP (1) with different f look very similar.

In two dimensions we worked with the following two functions as right hand side,

$$f_3(x) := 0, \tag{14}$$

$$f_4(x) := \sin(\pi x) \sin(\pi y). \tag{15}$$

The FEM results are presented in Table A.2.

B. PINN

In our experiments considering the PINN, we measure the error again in the L^2 and L^∞ norms. The L^2 norm is approximated using simple Monte Carlo integration, i.e.,

$$\|v\|^2_{L^2(\Omega \times [0,T])} \approx \frac{T \lambda^d(\Omega)}{n^{d+1}} \sum_{i=1}^{n} \sum_{j=1}^{n^d} v(t_i, \mathbf{x}_j)^2$$

where λ^d denotes the d-dimensional Lebesgue measure. Here, we use n equidistant grid points in each dimension, i.e., n^{d+1} in total, to test the numeric solution against the analytic one. The L^∞ is approximated by evaluating (13) on the same mesh.

A heavily used optimizer for neural networks is Adam and we want to analyze its performance on our PINNs. For this analysis we use the one dimensional heat equation with right hand side f_2 and a PINN with layers 32,

Learning rate	L^2 error	L^∞ error
0.0001	0.21534	1.36595
0.001	0.21054	1.33114
0.01	0.14159	0.79155
0.1	0.20866	0.78302
1	0.20866	0.78302

Table A.3. Test L^2 and L^∞ errors for different learning rates.

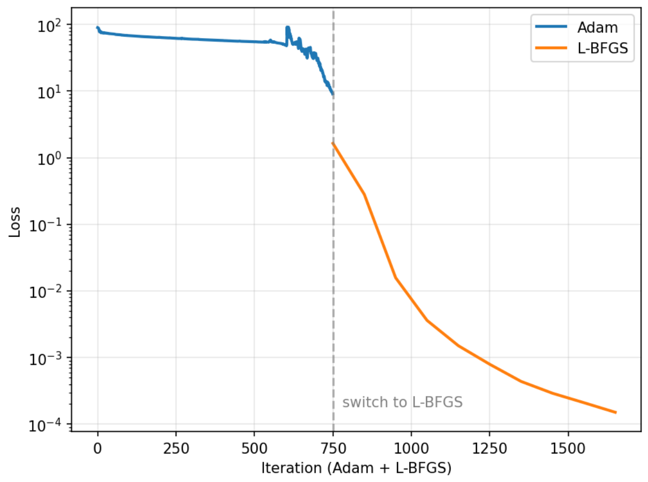


Figure 2. Progress of the training loss during training with Adam and L-BFGS

64, 128, 128 and tanh as activation functions. The results for 1000 Adam epochs with different learning rates are presented in table A.3. At first, we can see that a learning rate of 0.01 leads to the best results. But the errors are quite high and the second insight is that Adam alone is not sufficient to estimate the PDE on a good level.

We can further improve our optimization. As explained in our previous section the time scaling depends on a constant. We can optimize this constant during our weight update steps of the nodes, too. This can force the residuals to stay in the same range for all time points and improves Adam. It has the drawback that the time scale can explode and sometimes multiple random starts were needed. The most important enhancement is to use a more sophisticated optimizer that takes the second derivative into account, because the optimization problem created by the PINN is very complex. As L-BFGS is very sensitive to the initialization it was good practice to perform multiple epochs with Adam and use these results as starting point for L-BFGS.

Figure 2 visualizes these effects. Here we trained the same PINN as before but with optimizing the time scaling and 10 L-BFGS epochs with maximum 100 iterations after 750 Adam epochs. We plot on the y-axis the training loss and on the x axis we plot the number of iterations, where one Adam epoch has one iteration. For simplification we assume that L-BFGS uses in every epoch the maximum of 100 iterations. Adam has a very slow

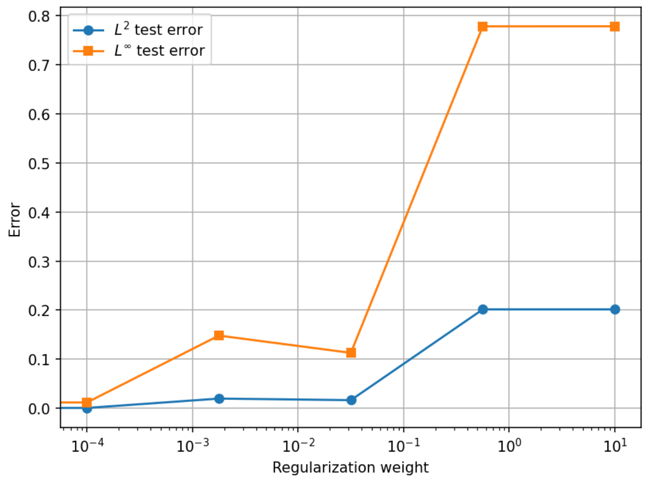


Figure 3. Test L^2 and L^∞ errors for different λ using L^2 regularization.

convergence until (perhaps due to a change in the time scale) it suddenly speeds up. The main loss reduction happens with L-BFGS which has a way better rate of convergence.

In the following, we want to compare the activation functions tanh, ReLU and sigmoid and have a look at the one dimensional heat equation with zero right hand side. We fit a neural network with four hidden layers each 64 nodes without optimizing the time scaling. At first, we fit it with 750 Adam epochs and evaluate it. Afterwards, we take the pretrained network and do 10 epochs of L-BFGS with 100 iterations. As discussed before L-BFGS sometimes does not find a solution. Thus, we repeat a trial if the L^2 error is greater than 10 but at most three times. The results are shown in table A.4. For this PDE a neural network with Adam alone produces reasonable results, whereas the networks with ReLU and sigmoid can not fit the PDE properly. Sigmoid performs slightly better than ReLU. Looking at the performance after the additional L-BFGS epochs amplifies these contrasts. The network with tanh improves a lot due to L-BFGS. With sigmoid at least the L^∞ error can be improved, but also this fit is not decent. For ReLU the L-BFGS optimization fails in three consecutive trials and outputs nan. This shows clearly that tanh is the right choice for our PINNs.

Metric / Activation	tanh	ReLU	sigmoid
L^2 (Adam)	0.013512	0.213668	0.181448
L^∞ (Adam)	0.095862	1.752794	1.039341
L^2 (L-BFGS)	0.000028	nan	0.199416
L^∞ (L-BFGS)	0.000174	nan	0.775914

Table A.4. Test L^2 and L^∞ errors for different activation functions using Adam and L-BFGS optimizers.

Regularization improves many neural networks and we test λ from 0.0001 until 10 for L^2 regularization. As setup we use a neural network with tanh layers 32, 64, 128, 128 to solve the one dimensional heat equation with zero right hand side. We use Adam and L-BFGS with the same number of iterations as before and also optimize the time scale. The results are shown in Figure 3. We can see that a higher λ yields to a higher error and the neural networks performs best without L^2 regularization. The reason for this is that we do not have any noise in the data and L-BFGS excels at finding the exact solution. As these results are clear we do not expect different results for L^1 regularization and do not use any regularization in the following.

Layers	L^2 error	L^∞ error	runtime
128, 128, 64, 32	0.001137	0.002633	323.645
32, 64, 128, 128	0.001869	0.004968	423.081
32, 64	0.003116	0.013494	92.444
32, 64, 64	0.508610	0.845356	138.729
64, 64, 32	nan	nan	–
128, 64, 32	nan	nan	–
128, 64	0.003581	0.012243	161.091
128, 128, 64, 64, 32	0.000485	0.001214	400.534
128, 128	0.003409	0.008391	269.763
128, 128, 128	0.000756	0.002190	407.261
128, 128, 128, 128	0.000055	0.000448	481.444
256, 256	0.000240	0.002138	603.237
512	0.004115	0.042084	490.854

Table A.5. Test of different Layer sizes for PINNS solving the one-dimensional heat equation

In another numerical experiment we tested different network architectures, i.e., networks with different numbers of hidden layers and nodes per layer. For the experiments we considered the one-dimensional problem with the right hand side given by f_2 in (11) and always tanh as activation function. In each example, 400 epochs with Adam and 10 epochs with L-BFGS were performed. Table A.5 shows the results of the experiments. Here, we see that a larger number of nodes overall leads to lower L^2 and L^∞ errors, as expected. With our best model we achieve an L^2 error of approximately $5 \cdot 10^{-5}$ and an L^∞ error of approximately $4 \cdot 10^{-4}$. This comes at the cost of a long computation time, which is about 480 seconds. Comparing the last three lines in Table A.5, we observe that deep networks are probably better than wide networks. Also, if the number of layers and nodes is the same in different PINNs, but the order of the layers is different, the results do not differ significantly.

Moreover, even with a simple model with two hidden layers and 32 and 64 nodes, respectively, the L^2 error is already approximately $3 \cdot 10^{-3}$ but computing the solution barely lasts 92 seconds.

Another finding is that the results tend to be slightly better when the number of nodes decreases to the end of the network. However, the differences are not significant. Thus, the number of parameters seems to be more important than the order of the different-sized layers. In two of the examples we obtained an error and the PINN did not yield a solution to the PDE.

right hand side	L^2 error	L^∞ error	run time [s]
f_1	0.000158	0.001053	458.275
f_2	$6.257 \cdot 10^{-5}$	0.000403	523.314

Table A.6. Results for our PINN in 1D

right hand side	L^2 error	L^∞ error	run time [s]
f_3	0.014392	0.371893	3,081.728
f_4	0.001697	0.042812	3,065.578

Table A.7. Results for our PINN in 2D

Figure 4 visualizes an example for a solution obtained by our PINN, in this case with 128, 128, 64 and 32 nodes in the four hidden layers. The analytic solution is plotted as a reference.

Finally, we used one of our best model from Table A.5 to compute solutions to the PDE in one and two dimension with the right hand sides given by (10), (11), (14) and (15). Here, we used a PINN with five hidden layers with 128, 128, 64, 64 and 32 layers, since the results in previous experiments were promising and the run time was not too high. For the 1D problem, we ran 400 epochs with Adam (learning rate 10^{-2}) and 10 epochs à 50 iterations with L-BFGS (learning rate 1). The results are shown in Table A.6. In the two-dimensional case, with right hand side f_3, we chose the same network architecture and parameters. But the number of epochs was increased to 500 for Adam and 20 for L-BFGS. Table A.7 show the results for these experiments.

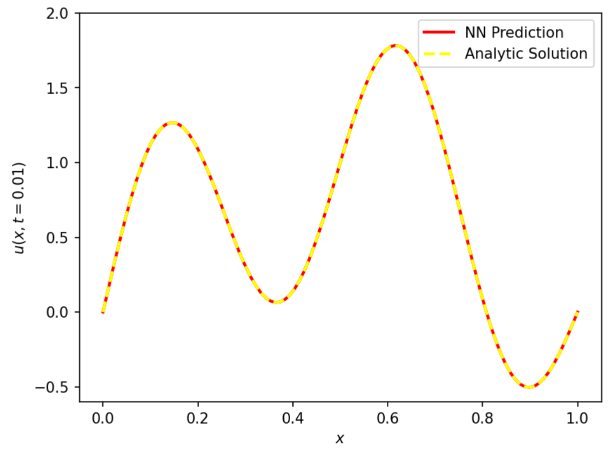
Figures 5 and 6 show the analytic and the PINN solution, respectively, at times $t = 0.01$ and $t = 0.1$, with the right hand side f in (1) given by $f_3 \equiv 0$.

In the one-dimensional examples we achieve errors that are much better than those we had with FEM. In one dimension, we reach L^2 errors between 10^{-5} and 10^{-4} with our best models and L^∞ errors between 10^{-3} and 10^{-4}. In two dimensions we still achieve L^2 errors around 10^{-3} to 10^{-2} and L^∞ errors of order 10^{-2} or 10^{-1}. If the right hand side of equation (1) is given by f_4 in (15), this even beats our implementation of FEM.

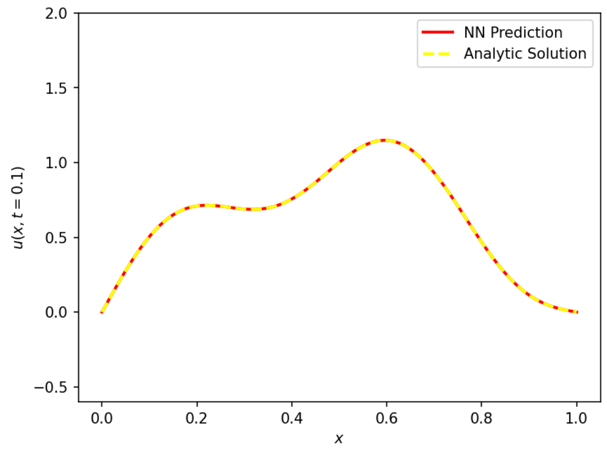
However, the runtime is very high in all examples. For one-dimensional problems we needed approximately 500 seconds to compute a solution, for two-dimensional problems even 3,000 seconds, i.e., almost an hour.

VI. CONCLUSION

Our PINN achieves good results for both, the one- and the two-dimensional heat equation. Applying our best models to the given problem, we could even reach

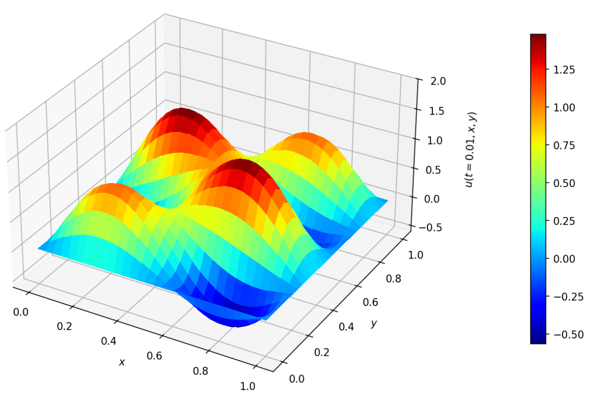


(a) At time $t = 0.01$

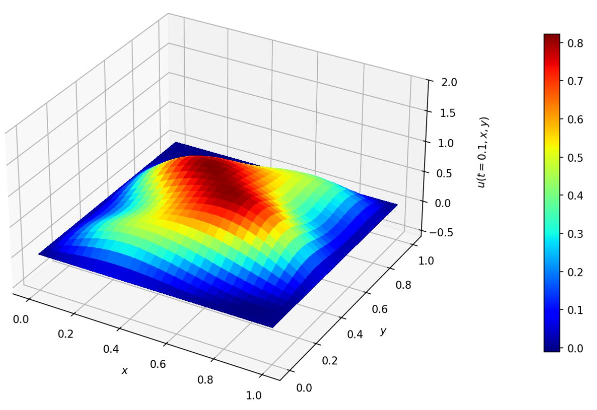


(b) At time $t = 0.1$

Figure 4. 1D PINN solution with right-hand side $f(x) = \sin(\pi x)$



(a) At time $t = 0.01$



(b) At time $t = 0.1$

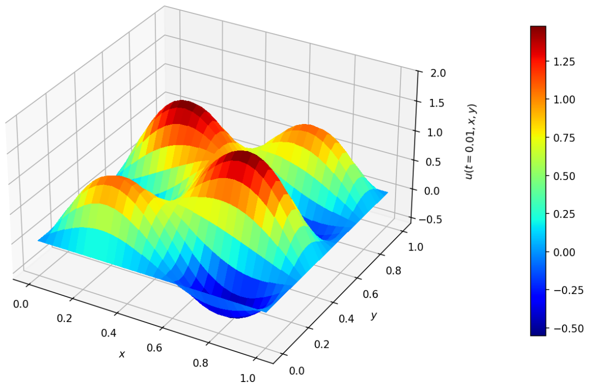
Figure 5. 2D PINN solution with right-hand side $f(x, y) = 0$

smaller errors than with the simple FEM implementation. Hence, our results show that neural networks are useful tools to solve PDEs numerically.

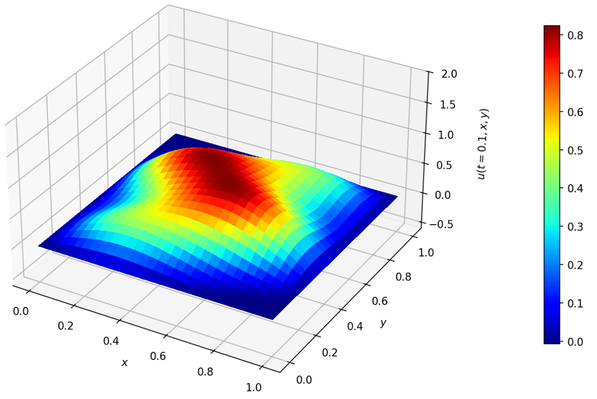
The big drawback of the PINN is its long runtime which is around 150 times higher then for the FEM examples. In fact, one can not reduce the runtime significantly by reducing the number of epochs in the training since this would result in bad approximations.

While we saw in our experiments that PINNs are compatible with FEM in low dimensions, previous work shows that neural networks can be particularly beneficial for high-dimensional problems [26]. In this case, the network architecture should be improved and we assume that our PINN would not yield good results in high dimensions. Moreover, on the one hand, no great mathematical knowledge is required to set up a PINN, in contrast to FEM where one needs to derive the weak formulation of the PDE problem. On the other hand, it is of course helpful to have some theoretical background when tuning parameters of the PINN. In our case, we chose the initial trial solution in a suitable way, exploiting knowledge about the exponential decay of solutions to the heat equation. If the trial solution was chosen at random, it would impair the training and final result.

The results obtained by FEM are satisfying, but can of course be much better if more work is done on parameter tuning. An easy improvement is to increase the number of grid points. However, we wanted to achieve a fair comparison between our PINN and FEM and therefore used a comparable number of grid points in both settings. Next, finite elements of higher degrees can be implemented [6, Sc. 3.2]. Moreover, an adaptive mesh [6, Ch. 9] could also improve the approximation. In combination with the short computation time it is therefore clear that par-



(a) At time $t = 0.01$



(b) At time $t = 0.1$

Figure 6. Analytic solution in 2D with right-hand side $f(x, y) = 0$

ticularly for low-dimensional problems, the finite element method should be preferred over neural networks. For a suitable comparison on high-dimensional problem more research is required.

[1] L. C. Evans, *Partial differential equations* (American Mathematical Society, Providence, R.I., 2010).
[2] H. Brezis, *Functional analysis, Sobolev spaces and partial differential equations*, Vol. 2 (Springer, 2011).
[3] M. Dobrowolski, Angewandte funktionalanalysis: Funktionalanalysis, sobolev-räume und elliptische differentialgleichungen (Springer, 2006) Chap. 7.7.
[4] R. Haberman, *Elementary applied partial differential equations*, Vol. 987 (Prentice Hall Englewood Cliffs, NJ, 1983).
[5] A. Durmagambetov and L. Fazilova, Navier-stokes equations—millennium prize problems, Natural Science **7**, 88 (2015).
[6] S. C. Brenner and L. R. Scott, *The mathematical theory of finite element methods* (Springer, 2008).
[7] D. Braess, *Finite elements: Theory, fast solvers, and applications in solid mechanics* (Cambridge University Press, 2001).
[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations (2017), arXiv:1711.10561 [cs.AI].
[9] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations (2017), arXiv:1711.10566 [cs.AI].
[10] J. Blechschmidt and O. G. Ernst, Three ways to solve partial differential equations with neural networks – a review (2021), arXiv:2102.11802 [math.NA].
[11] H.-J. Bungartz, S. Zimmer, M. Buchholz, and D. Pflüger, Modeling and simulation: an application-oriented introduction (Springer Science & Business Media, 2013) Chap. 14.
[12] A. Logg, K.-A. Mardal, G. N. Wells, *et al.*, *Automated Solution of Differential Equations by the Finite Element Method* (Springer, 2012).
[13] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, Automatic differentiation in machine learning: a survey, Journal of machine learning research **18**, 1 (2018).
[14] K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, Neural networks **2**, 359 (1989).
[15] I. A. Baratta, J. P. Dean, J. S. Dokken, M. Habera, J. S. Hale, C. N. Richardson, M. E. Rognes, M. W. Scroggs, N. Sime, and G. N. Wells, DOLFINx: the next generation FEniCS problem solving environment, preprint (2023).
[16] M. S. Alnaes, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, Unified form language: A domain-specific language for weak formulations of partial differential equations, ACM Transactions on Mathematical Software **40**, 10.1145/2566630 (2014).
[17] M. W. Scroggs, I. A. Baratta, C. N. Richardson, and G. N. Wells, Basix: a runtime finite element basis evaluation library, Journal of Open Source Software **7**, 3982 (2022).
[18] M. W. Scroggs, J. S. Dokken, C. N. Richardson, and G. N. Wells, Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes, ACM Transactions on Mathematical Software **48**, 18:1 (2022).
[19] J. Stoer and R. Bulirsch, Numerische mathematik 2: Eine einführung—unter berücksichtigung von vorlesungen von fl bauer (Springer, 2005) pp. 311–320.
[20] B. Sullivan and A. Kaszynski, Pyvista: 3d plotting and mesh analysis through vtk, Journal of Open Source Software (2019).
[21] https://github.com/thomas-engl/Project-3-Applied-ML/tree/main/Animation.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, Pytorch: An imperative style, high-performance deep learning library (2019), arXiv:1912.01703 [cs.LG].
[23] D. C. Liu and J. Nocedal, On the limited memory bfgs method for large scale optimization, Mathematical programming **45**, 503 (1989).
[24] F. Hasan, H. Ali, and H. A. Arief, From mesh to neural nets: A multi-method evaluation of physics informed neural network and galerkin finite element method for solving nonlinear convection–reaction–diffusion equations, International Journal of Applied and Computational Mathematics **11**, 1 (2025).
[25] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer Series in Operations Research and Financial Engineering (Springer, 2006).
[26] Z. Hu, K. Shukla, G. E. Karniadakis, and K. Kawaguchi, Tackling the curse of dimensionality with physics-informed neural networks, arXiv preprint (2023).

Appendix A: Proofs

Proof of Theorem 1. We need to show that a is a continuous and coercive bilinear form and that ℓ is linear and bounded. Then, the result follows from the Lax-Milgram lemma (see, e.g. [1, Sc 6.2.1]).

The bilinearity of a is clear. For $u, v \in H_0^1(\Omega)$, we have, by Hölder's inequality,

$$\begin{aligned}|a(u,v)| &\leq \int_\Omega |uv| + h\kappa|\nabla u||\nabla v| \ \mathrm{d}\mathbf{x} \\ &\leq \|u\|_{L^2}\|v\|_{L^2} + h\kappa\|\nabla u\|_{L^2}\|\nabla v\|_{L^2} \\ &\leq C_1\|u\|_{H^1}\|v\|_{H^1}\end{aligned}$$

where $C_1 := \max\{1, h\kappa\}$. Hence, a is continuous. Coercivity follows from

$$\begin{aligned}a(u,u) &= \int_\Omega |u|^2 + h\kappa|\nabla u|^2 \ \mathrm{d}\mathbf{x} \\ &\geq C_2\|u\|_{H^1}^2\end{aligned}$$

for all $u \in H_0^1(\Omega)$ with $C_2 := \min\{1, \kappa h\} > 0$.

ℓ is clearly a linear form on $H_0^1(\Omega)$. It is bounded since

$$\begin{aligned}|\ell(v)| &\leq h\int_\Omega |fv| \ \mathrm{d}\mathbf{x} + \int_\Omega |u^{(i)}v| \ \mathrm{d}\mathbf{x} \\ &\leq h\|f\|_{L^2}\|v\|_{L^2} + \|u^{(i)}\|_{L^2}\|v\|_{L^2} \\ &\leq C_3\|v\|_{L^2} \\ &\leq C_3\|v\|_{H^1}\end{aligned}$$

for all $v \in H_0^1(\Omega)$ where the constant is $C_3 := \max\{h\|f\|_{L^2}, \|u^{(i)}\|_{L^2}\}$. □

Proof of Proposition 1. We present the proof for the one-dimensional case. The procedure for the two-dimensional problem is similar but more technical. Since we work with Lagrange-elements of degree 1, the basis functions of the finite-dimensional subspace are hat functions ϕ_i, $i = 1, ..., n$ where n is the number of inner grid points x_i. Those functions satisfy $\phi_i(x_i) = \delta_{ij}$ and on $[x_{i-1}, x_{i+1}]$ they are piecewise linear.

Let h be the mesh size, i.e., $h = x_i - x_{i-1}$ for all i. We assemble the system matrix [7, Ch. 8.1]

$$A = (a_ij)_{i,j=1}^n \quad \text{with} \quad a_{ij} = a(\phi_i, \phi_j)$$

where a is the bilinear form from (8).

Note that if $|i-j| \geq 2$, $\operatorname{supp}(\phi_i\phi_j) = \emptyset$. If $|i-j| = 1$, then $\operatorname{supp}(\phi_i\phi_j) = [\min\{x_i, x_j\}, \max\{x_i, x_j\}]$ and if $i = j$, then $\operatorname{supp}(\phi_i^2) = [x_{i-1}, x_{i+1}]$. It is easy to see that we can compute the integrals on the reference element defined on the interval $[0, 1/h]$. It holds

$$\begin{aligned}\int_0^1 \phi_i^2 + \tau\kappa(\phi_i')^2 \ \mathrm{d}x &= \int_{x_{i-1}}^{x_{i+1}} \phi_i^2 + \tau\kappa(\phi_i')^2 \ \mathrm{d}x \\ &= 2\int_0^h \frac{1}{h^2}x^2 + \tau\kappa\frac{2}{h^2} \ \mathrm{d}x \\ &= \frac{2}{3}h + \frac{4\tau\kappa}{h}.\end{aligned}$$

If $i \neq j$, $a_{ij} = 0$ as soon as $|i-j| \geq 2$. In the other case,

$$\begin{aligned}\int_0^1 \phi_i\phi_j + \tau\kappa\phi_i'\phi_j' \ \mathrm{d}x &= \int_0^h \frac{1}{h^2}x(h-x) - \tau\kappa\frac{1}{h^2} \ \mathrm{d}x \\ &= \frac{h}{6} - \frac{\tau\kappa}{h}.\end{aligned}$$

Thus, the system matrix is given by

$$A = \operatorname{tridiag}\left(\frac{h}{6} - \frac{\tau\kappa}{h}, \frac{2h}{3} + \frac{4\tau\kappa}{h}, \frac{h}{6} - \frac{\tau\kappa}{h}\right).$$

Hence, A is symmetric. Moreover, for all $i = 1, ..., n$, we have

$$\begin{aligned}\sum_{j\neq i} |a_{ij}| &\leq 2\left|\frac{h}{6} - \frac{\tau\kappa}{h}\right| \\ &\leq \frac{h}{3} + \frac{2\tau\kappa}{h} \\ &< a_{ii}.\end{aligned}$$

Thus, by Gershgorin's circle theorem, all eigenvalues are positive, i.e., A is positive definite. □