Physics Informed Neural Networks for Non-Stationary Material Science Problems

Paweł Maczuga[0000−0002−5111−6981], Tomasz Służalec[0000−0001−6217−4274], Łukasz Sztangret[0000−0003−4872−406X], Danuta Szeliga[0000−0002−2915−8317], Marcin Łoś[0000−0002−8426−6345] and Maciej Paszyński[0000−0001−7766−6052]

AGH University of Krakow, Poland
{pmaczuga,sluzalec,szt,szeliga,los,paszynsk}@agh.edu.pl

Abstract. Linear elasticity and Navier-Stokes equations are fundamental tools in material science, enabling the modeling of solid deformations and fluid flows under various conditions. These equations are widely used to simulate stresses, strains, and fluid interactions in processes like 3D printing, welding, casting, and extrusion. Physics-Informed Neural Networks (PINNs), introduced in 2019, have gained significant attention for solving complex physical problems, including fluid mechanics, wave propagation, and inverse problems. Despite their growing popularity, PINNs face challenges in training efficiency and accuracy. This paper investigates the applicability of modern PINN methodologies to material science problems involving Navier-Stokes and linear elasticity equations. For linear elasticity, a randomized selection of collocation points is employed to enhance training. For Navier-Stokes equations, hard constraints on initial and boundary conditions are implemented to avoid multi-objective optimization. These approaches aim to address training difficulties and improve PINN performance in simulating material science phenomena.

Keywords: Physics Informed Neural Networks, Transient linear elasticity, Navier-Stokes, Random selection of collocation points, Hard constraints

1 Introduction

Linear elasticity [21] and Navier-Stokes equations [35] are foundational in simulating various physical phenomena in material science. They are used to model the behavior of fluids and solids, respectively, and help researchers and engineers understand and design materials under different conditions. Linear elasticity equations describe the deformation of solid materials under applied loads. These equations are widely used in material science to model mechanical behavior. They are used to predict the distribution of stresses and strains within a solid material under mechanical load, to understand material strength, durability, and failure mechanisms. They are employed to simulate residual stresses and distortions that arise during manufacturing processes like welding or 3D printing. The Navier-Stokes equations describe the motion of fluids and are essential

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

for studying fluid flow behavior. In material science, they are used to model how fluids flow through porous materials such as rocks, membranes, or composite materials. They can be combined with elasticity equations to simulate how a fluid interacts with a deformable solid boundary. They can also be employed to simulate fluid behavior in material science processes like casting, extrusion, and additive manufacturing, to optimize manufacturing processes, improve material properties, and reduce defects.

The family of Physics Informed Neural Network (PINN) solvers have been introduced by prof. George Karniadakis in 2019 [30]. The method gained exponential growth in the number of papers and citations, with several new papers and some modifications of the method introduced every year [39,13,29]. The method, however, has some difficulties of the training process, some of them discussed in [37,32]. PINNs have been successfully applied to solve a wide range of problems, from fluid mechanics [3,20], in particular Navier-Stokes equations [16,34,36], wave propagation [31,19,7], phase-filled modeling [9], biomechanics [2,15], quantum mechanics [12], electrical engineering [23], problems with point singularities [11], uncertainty qualification [38], dynamic systems [33,1], or inverse problems [4,22,18], among many others.

In this paper, we focus on the investigation of the applicability of the modern PINN methodology for modeling the Navier-Stokes and linear elasticity material science problems. In the linear elasticity method, we employ the randomized selection of the PINN colocation points to improve the training procedure. In the Navier-Stokes formulation, we employ the initial and boundary conditions using the hard constraints, avoiding a multi-objective optimization problem.

Our library is a simple alternative for other available PINN libraries, including the DeepXDE [17] and IDRLnet [28]. The first one is a very large library including ODEs and PDEs on complex geometries, with different initial and boundary conditions, enabling solving and forward and inverse problems. The DeepXDE supports TensorFlow, PyTorch, JAX, and PaddlePaddle. The second one allows for solving the wave equation, Allan-Cahn equations, Volterra integrodifferential equations, and variational minimization problems. The IDRLnet library uses pytorch, numpy, and Matplotlib.

2 Linear elasticity

We start from the following time-dependent linear elasticity equations:

$$\begin{cases} \rho \frac{\partial^2 \boldsymbol{u}}{\partial t^2} = \nabla \cdot \boldsymbol{\sigma} + \mathbf{F} & \text{on } \Omega \times [0,T] \\ \boldsymbol{u}(x,0) = u_0 & \text{for } x \in \Omega \\ \boldsymbol{\sigma} \cdot \hat{\boldsymbol{n}} = 0 & \text{on } \partial\, \Omega \end{cases} \tag{1}$$

where $\boldsymbol{u}$ is a displacement vector, $\sigma_{ij} = c_{ijkl}\epsilon_{lk} = \sum_{kl} c_{ijkl} \frac{1}{2}(\partial_j u_i + \partial_i u_j)$ is the stres tensor, $\epsilon_{lk} = \frac{1}{2}\left(\frac{\partial u_k}{\partial x_l} + \frac{\partial u_l}{\partial x_k}\right)$ is the strain tensor, $\boldsymbol{c}$ is a matrix describing the material properties. We assume that the computational domain is the cuboid $\Omega = [0,1] \times [0,1] \times [0,5]$, namely $x \in [0,1]$, $y \in [0,1]$ and $t \in [0,5]$.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

For our simulation, we employ the 2D version of the problem: We seek the displacement vector field $\Omega \times [0,T] \ni (x,y;t) \to \mathbf{u}(x,y;t) = (u_1(x,y;t), u_2(x,y;t)) \in R^2$, where for $i,j \in \{0,1\}$ we have

$$\begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix} = \begin{bmatrix} \lambda\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}\right) & 0 \\ 0 & \lambda\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}\right) \end{bmatrix} + 2\mu \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) \\ \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) & \frac{\partial u_y}{\partial y} \end{bmatrix} \tag{2}$$

and the two-dimensional linear elasticity equations read

$$\begin{cases} \rho \frac{\partial^2 u_x}{\partial t^2} = \frac{\partial \sigma_{11}}{\partial x} + \frac{\partial \sigma_{12}}{\partial y} + f_1 \\ \rho \frac{\partial^2 u_y}{\partial t^2} = \frac{\partial \sigma_{21}}{\partial x} + \frac{\partial \sigma_{22}}{\partial y} + f_2 \end{cases} \tag{3}$$

where we have assumed the forcing acting on the right-top corner

$$\begin{gathered} f_1(x,y,t) = -t^2(1-t)^2 r(x_1,x_2),\ f_2(x,y,t) = -t^2(1-t)^2 r(x_1,x_2), \\ r(x,y) = 10\exp\left(-10\left[(x_1-1)^2 + (x_2-1)^2\right]\right). \end{gathered} \tag{4}$$

We introduce the following boundary conditions

$$\begin{cases} \boldsymbol{u}(x_1,x_2,0) = u_0 = 0 \\ \boldsymbol{\sigma} \cdot \hat{\boldsymbol{n}} = 0 \end{cases} \tag{5}$$

The Lame coefficients in our model problem are defined as $\lambda = \mu = 1$.

3 Physics Informed Neural Networks for linear elasticity

For the solution of the linear elasticity problem by using the Physics Informed Neural Networks, we assume 4 layers of the neural network with 200 neurons in each layer. The input to the neural network is (x,y,t), and the output from the neural network is a vector $(u_x, u_y, \frac{\partial u_x}{\partial x}, \frac{\partial u_x}{\partial y}, \frac{\partial u_x}{\partial t}, \frac{\partial u_y}{\partial x}, \frac{\partial u_y}{\partial y}, \frac{\partial u_y}{\partial t},)$. The initial configuration is set as a hard constraint, that is, it is enforced in the network architecture. The initial condition is simple: $\boldsymbol{u}(x_1,x_2,0) = u_0 = 0$ So we have:

$$\begin{cases} u_x = u_{x_{original}} \cdot t \\ u_y = u_{y_{original}} \cdot t \end{cases} \tag{6}$$

We can see that for $t = 0$, u will always be 0, and for $t > 0$, u will also be > 0, so it can be normally influenced by the neural network parameters. The loss function is divided into two parts, $LOSS_{PDE}$ and $LOSS_{boundary}$. $LOSS_{PDE}$ consists of 8 parts. The most important are the first two, the residuals of the system of equations (3) squared. The remaining 6 are derivatives of u_x and u_y

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

with respect to all inputs minus the corresponding output of the network. It is possible to compute the second derivative in the residual directly, in which case the network would only have 2 outputs. However, we found that having extra outputs with first derivatives greatly improves PINN's convergence.

For training, we used the Adam optimizer [14] with the learning rate of 0.005. We run the training on 40×40 points grid with equally distributed collocation points for 60 time steps and 20 000 epochs. In order to train the neural network, we need to create a loss function such that minimizing it will push PINN towards the solution of the equation (3). The loss is divided into two parts: PDE residual loss of equation (3), and the boundary loss for the boundary conditions. Initial conditions are set using the hard constraints as mentioned earlier. The PDE residual loss is further divided into eight parts, where the first two are the most important and the rest is introduced as a way to improve the convergence.

$$\text{LOSS}_{\text{PDE}} = \sum_{i=1}^{8} \text{LOSS}_{\text{PDE}_i} \tag{7}$$

$$\begin{aligned}
\text{LOSS}_{\text{PDE}_1} &= \left(\mu \frac{\partial \,\text{PINN}_5}{\partial t} - \frac{\partial \sigma_{11}}{\partial x_1} - \frac{\partial \sigma_{12}}{\partial x_2} + f_1\right)^2, \\
\text{LOSS}_{\text{PDE}_2} &= \left(\mu \frac{\partial \,\text{PINN}_8}{\partial t} - \frac{\partial \sigma_{11}}{\partial x_1} - \frac{\partial \sigma_{12}}{\partial x_2} + f_2\right)^2, \\
\text{LOSS}_{\text{PDE}_3} &= \left(\frac{\partial \,\text{PINN}_1}{\partial x} - \text{PINN}_3\right)^2, \quad \text{LOSS}_{\text{PDE}_4} = \left(\frac{\partial \,\text{PINN}_1}{\partial y} - \text{PINN}_4\right)^2, \\
\text{LOSS}_{\text{PDE}_5} &= \left(\frac{\partial \,\text{PINN}_1}{\partial t} - \text{PINN}_5\right)^2, \quad \text{LOSS}_{\text{PDE}_6} = \left(\frac{\partial \,\text{PINN}_2}{\partial x} - \text{PINN}_6\right)^2, \\
\text{LOSS}_{\text{PDE}_7} &= \left(\frac{\partial \,\text{PINN}_2}{\partial y} - \text{PINN}_7\right)^2, \quad \text{LOSS}_{\text{PDE}_8} = \left(\frac{\partial \,\text{PINN}_2}{\partial t} - \text{PINN}_8\right)^2.
\end{aligned} \tag{8}$$

where

$$\begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix} = \begin{bmatrix} \lambda \cdot (\text{PINN}_3 + \text{PINN}_7) + 2\mu \cdot \text{PINN}_3 & \mu \cdot (\text{PINN}_4 + \text{PINN}_6) \\ \mu \cdot (\text{PINN}_4 + \text{PINN}_6) & \lambda \cdot (\text{PINN}_3 + \text{PINN}_7) + 2\mu \cdot \text{PINN}_7 \end{bmatrix} \tag{9}$$

and f is the forcing term defined by defined by (4). Note that PINN_3 - PINN_8 are trained to be just derivatives of PINN_1 and PINN_2 with respect to all inputs. However, splitting it in this way proved to greatly improve the convergence. The loss responsible for enforcing boundary conditions is defined as follows:

$$\begin{aligned}
\text{LOSS}_{\text{boundary}} = &\,\text{LOSS}_{\text{boundary}_{down}} + \text{LOSS}_{\text{boundary}_{up}} \\
&+ \text{LOSS}_{\text{boundary}_{left}} + \text{LOSS}_{\text{boundary}_{right}}
\end{aligned} \tag{10}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

$$\begin{aligned} \mathrm{LOSS}_{\mathrm{boundary}_{down}} &= -\sigma_{12} - \sigma_{22}, \quad \mathrm{LOSS}_{\mathrm{boundary}_{up}} = \sigma_{12} + \sigma_{22}, \\ \mathrm{LOSS}_{\mathrm{boundary}_{left}} &= -\sigma_{11} - \sigma_{21}, \quad \mathrm{LOSS}_{\mathrm{boundary}_{right}} = \sigma_{11} + \sigma_{21}. \end{aligned} \tag{11}$$

4 Numerical results for linear elasticity

We have run the linear elasticity simulation for a time interval $[0, 5]$ over a cube-shaped domain $[0, 1]^2$. Notice that in the Physics Informed Neural Network simulations there are no time steps; the computational problem is solved over the space-time cuboid. In the initial configuration, the square-shaped body has fixed zero displacement. We hit the elastic body at the right top corner with the force defined by (4). The convergence of the training is presented in Figure 1. For the linear elasticity computations, we minimize two loss functions. The first loss function is the residual of the PDE (called the Loss PDE on the second panel in Figure 1). The second loss function is related with the minimization of the boundary condition (5) (called the Loss boundary on the third panel in Figure 1). We do not know how to enforce this kind of tensorial boundary condition as the hard constraint of the neural network. Thus, we minimize the sum of the two losses, as it is presented in the first panel in Figure 1.

The snapshots from the simulation are presented in Figure 2. We have multiplied the displacements by a factor of 10 for a better visualization of the results. However, with the central point fixed in the body, it is possible that small displacements around the central point will be visualized as moving to the other side of the central point. This, however, does not happen in reality; it is just a drawback of our visualization method.

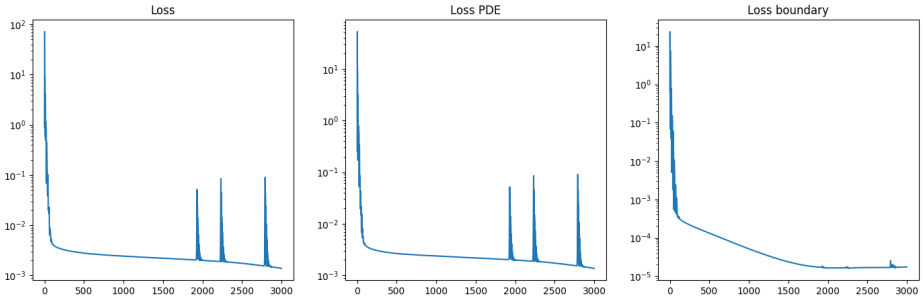

Fig. 1: The convergence of the training of the total loss function for the transient linear-elasticity problem. The total loss, the residual loss, and the boundary condition loss.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

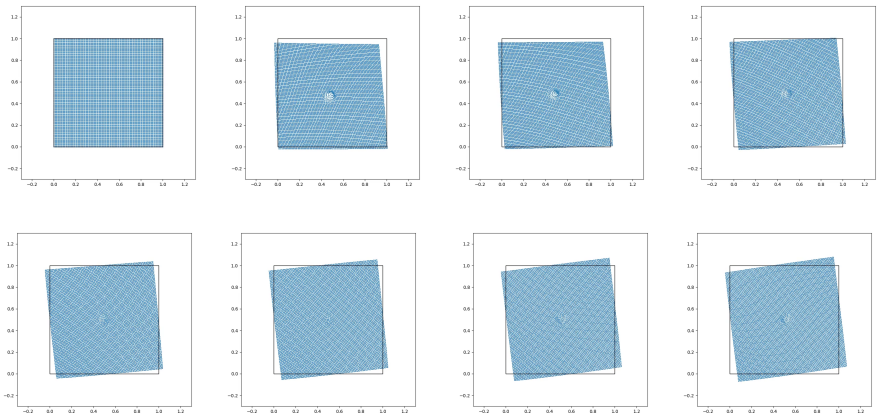

Fig. 2: Snapshots from the linear elasticity simulations. The body has fixed zero displacements in the central point.

5 Navier-Stokes equations

Let us focus on the non-stationary cavity flow problem described with the Navier-Stokes equation for the incompressible fluid; see Figure 3. The Dirichlet boundary condition drives the cavity flow for the velocity $u_x = 1$, $u_y = 0$ on the top boundary. On the remaining parts of the boundary, the velocity is equal to 0, and the ϵ thick transition zone in the left and right top corners ensures the possibility of a weak formulation. This problem exhibits pressure singularities at the two corners.

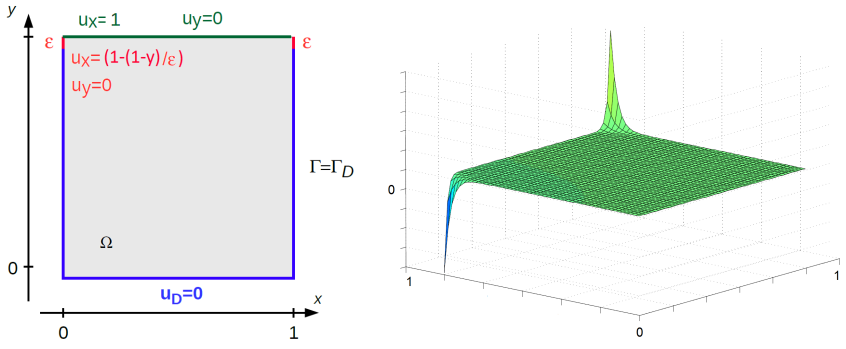

Fig. 3: Non-stationary cavity flow problem. Boundary conditions and pressure singularities.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

Let $\Omega = (0,1)^2$ be the open boundary and $I = [0,T] \subset \mathcal{R}$ be the time interval. The problem reads: Find velocity u and pressure field p such that:

$$\begin{cases} \partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} - \Delta\mathbf{u} + \nabla p = 0 & \text{in } \Omega \times I, \\ \nabla\cdot\mathbf{u} = 0 & \text{in } \Omega \times I, \\ \mathbf{u} = h & \text{in } \Gamma \times I, \\ \mathbf{u}(0) = 0 & \text{in } \Omega, \end{cases} \tag{12}$$

where

$$h(x,y) = \begin{cases} 0 & x \in (0,1),\ y = 0 \\ 0 & x \in \{0,1\},\ y \in (0, 1-\epsilon) \\ 1 & x \in (0,1),\ y = 1 \\ \left(1 - \dfrac{(1-y)}{\epsilon}\right) & x \in \{0,1\},\ y \in (1-\epsilon, 1) \end{cases} \tag{13}$$

By incorporating a shift

$$\mathbf{u}_D = \left(\begin{cases} \left(1 - \dfrac{(1-y)}{\epsilon}\right)^2 & x \in (0,1),\ y \in (1-\epsilon, 1) \\ 0 & x \in (0,1),\ y \in (0, 1-\epsilon) \end{cases}, \ 0 \right) \tag{14}$$

this problem transforms into

$$\begin{cases} \partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} - \Delta\mathbf{u} + \nabla p = \mathbf{f} & \text{in } \Omega \times I, \\ \nabla\cdot\mathbf{u} = 0 & \text{in } \Omega \times I, \\ \mathbf{u} = 0 & \text{in } \Gamma \times I, \\ \mathbf{u}(0) = 0 & \text{in } \Omega, \end{cases} \tag{15}$$

where $\mathbf{f} = \left(0, \frac{2}{\epsilon}\left(1 - \frac{(1-y)}{\epsilon}\right)^3\right)$. Here $\mathbf{u} = (u_1, u_2)$ represents the velocity vector field, and p represents the scalar pressure field. The Γ denotes the boundary of the spatial domain Ω, and $\mathbf{f}$ is a given source resulting from the shift of the Dirichlet boundary conditions.

System (15) can be rewritten as

$$\begin{aligned} w_1(x_1,x_2,t) = \frac{\partial u_1(x_1,x_2,t)}{\partial x_1}, \quad w_2(x_1,x_2,t) = \frac{\partial u_1(x_1,x_2,t)}{\partial x_2}, \\ z_1(x_1,x_2,t) = \frac{\partial u_2(x_1,x_2,t)}{\partial x_1}, \quad z_2(x_1,x_2,t) = \frac{\partial u_2(x_1,x_2,t)}{\partial x_2}, \\ -\frac{\partial w_1(x_1,x_2,t)}{\partial t} - \frac{\partial w_1(x_1,x_2,t)}{\partial x_1} - \frac{\partial w_2(x_1,x_2,t)}{\partial x_2} + \frac{\partial p(x_1,x_2,t)}{\partial x_1} = f_1(x_1,x_2,t), \\ -\frac{\partial z_1(x_1,x_2,t)}{\partial t} - \frac{\partial z_1(x_1,x_2,t)}{\partial x_1} - \frac{\partial z_2(x_1,x_2,t)}{\partial x_2} + \frac{\partial p(x_1,x_2,t)}{\partial x_2} = f_2(x_1,x_2,t), \\ \frac{\partial u_1(x_1,x_2,t)}{\partial x_1} + \frac{\partial u_2(x_1,x_2,t)}{\partial x_2} = 0. \end{aligned} \tag{16}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

We define the following residual functions

$$\begin{aligned} RES_{6a}(u_\theta) = \frac{\partial u_1}{\partial x_2} - w_2,\ RES_{6b}(u_\theta) = \frac{\partial u_2}{\partial x_1} - z_1,\ RES_{6c}(u_\theta) = \frac{\partial u_2}{\partial x_2} - z_2, \\ RES_{6d}(u_\theta) = \frac{\partial w_1}{\partial t} - \frac{\partial w_1}{\partial x_1} - \frac{\partial w_2}{\partial x_2} + \frac{\partial p}{\partial x_1} - f_1, \\ RES_{6e}(u_\theta) = \frac{\partial z_1}{\partial t} - \frac{\partial z_1}{\partial x_1} - \frac{\partial z_2}{\partial x_2} + \frac{\partial p}{\partial x_2} - f_2, \\ RES_{6f}(u{\cdot}\theta) = \frac{\partial u_1(x_1,x_2)}{\partial x_1} + \frac{\partial u_2(x_1,x_2)}{\partial x_2},\ RES_{6g}(u_\theta) = \frac{\partial u_1}{\partial x_1} - w_1. \end{aligned} \tag{17}$$

and the following total loss

$$\begin{aligned} RES(u_\theta) = RES_{6a}(u_\theta) + RES_{6b}(u_\theta) + RES_{6c}(u_\theta) + RES_{6d}(u_\theta) + \\ RES_{6e}(u_\theta) + RES_{6f}(u_\theta) + RES_{6g}(u_\theta). \end{aligned} \tag{18}$$

The Dirichlet boundary condition is obtained by multiplication of the output from the neural network by a summation of the four functions presented in Figure 4 multiplied by the **g** function (definition of the Dirichlet b.c.). For the pressure approximation, we multiply the output from the neural network by the single value of the solution at the central point, see Figure 4.

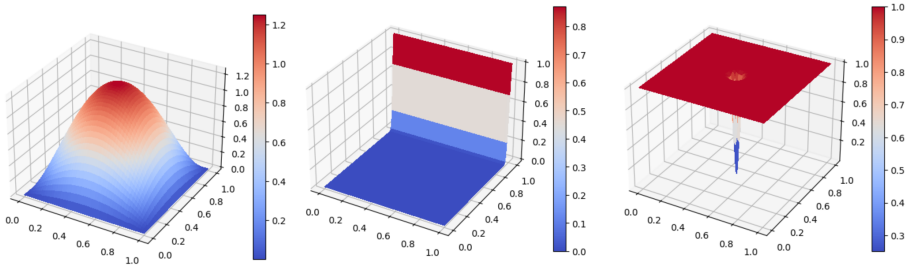

Fig. 4: The functions employed to enforce zero Dirichlet b.c., to enforce the boundary condition at the top of the domain, and the enforce zero pressure condition at the center of the domain.

6 Numerical results for transient Navier-Stokes

We have run the Navier-Stokes problem for the interval $[0, 1]$ over the cube-shaped cavity of dimensions $[0, 1]^2$. In the initial configuration, the velocity and the pressure inside the cavity are zero. The flow is driven by the boundary condition at the top edge of the computational domain, where the "river" flows from the left to the right. The convergence of the training is presented in Figure

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

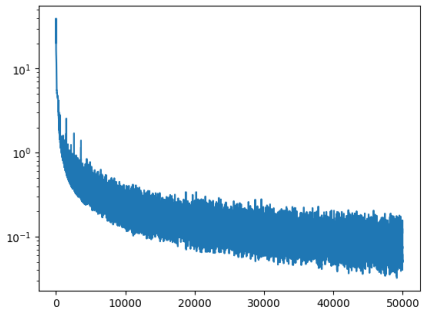

Fig. 5: The convergence of the training of the total loss function for the transient Navier-Stokes problem.

5. The boundary conditions are enforced by the hard constraint on the neural network, so there is no other loss function to minimize there.

The snapshots from the simulation are presented in Figure 6.

We compare our PINN code with IGA-FEM code [41]. For the IGA-FEM we remove the non-linear term $u \cdot \nabla u$ since it requires special linearization treatment. For the finite element method formulation, following [10], we consider the singular perturbation of non-stationary Navier-Stokes problem

$$\begin{cases} \partial_t \mathbf{v}_\epsilon - \Delta \mathbf{v}_\epsilon + \nabla p_\epsilon = \mathbf{f} & \text{in } \Omega \times I, \\ \epsilon A \phi_\epsilon + \nabla \cdot \mathbf{v}_\epsilon = 0 & \text{in } \Omega \times I, \\ \epsilon \partial_t p_\epsilon = \ \phi_\epsilon - \chi \nabla \cdot \mathbf{v}_\epsilon & \text{in } \Omega \times I, \\ \mathbf{v}_\epsilon = 0 & \text{in } \Gamma \times I, \\ \mathbf{v}_\epsilon(0) = \mathbf{v}_0 & \text{in } \Omega, \\ p_\epsilon(0) = p_0 & \text{in } \Omega, \end{cases} \tag{19}$$

where A is an unbounded operator $A : D(A) \subset L_0^2(\Omega) \longrightarrow L_0^2(\Omega)$ and $\phi_\epsilon \in D(A)$. Here, ϵ is the perturbation parameter and $\chi \in [0, 1]$ is a user-defined parameter. We consider the alternating directions method presented in [10] with the Peaceman-Rachford scheme applied to the velocity update. We employ the residual minimization method with B-spline basis functions for discretization. The resulting IGA-FEM (isogeometric finite element) method is summarized in [40]. Figure 6 presents the visual comparison of results using 80×80 mesh for the time moment $t = 0.5$. The execution time for the IGA-FEM code, depending on the B-spline basis function used for discretization (see Table 1 in [40]), varies from 300 seconds (5 minutes) to 2100 seconds (35 minutes), using 1024 time steps. The training time for our PINN code is around 15 minutes on A100 card from Google Colab.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

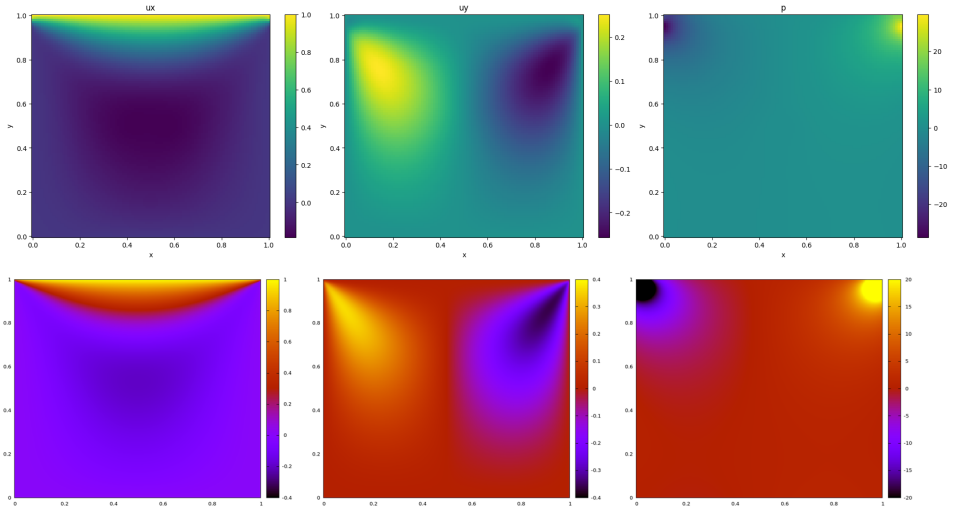

Fig. 6: The velocity component in the x direction, the velocity component in the y direction, and the pressure distribution p. **Top panel**: PINN solution extracted from the neural network solution at the time moment $t = 0.5$. **Bottom panel**: IGA-FEM solution.

7 Summary of the code

The code for both simulations: linear elasticity and Navier-Stokes are independent Jupyter Notebooks run in a Google Colab environment:

`https://colab.research.google.com/drive/1CxCbbMfS1C2y-Q1w_mWA6YR1706N6M-C`

`https://colab.research.google.com/drive/1lzq7qhlnIi5_Mz7b0PE0Fq_f958ANZlr`

The code has the following structure. The code is tuned with neural network parameters such as `LAYERS = 4`, `NEURONS_PER_LAYER = 200`, training parameters such as `LEARNING_RATE = 0.005`, and `EPOCHS = 20_000`, as well as `X_POINTS = 100` and `Y_POINTS = 100` defining the grid of training points. We also define the plotting parameters `X_PLOT = 100` and `Y_PLOT = 100` the accuracy of the graphics. There are two parts of the code that require modification when implementing new simulation. The first one is the residual loss function. For example, the loss function for the Navier-Stokes equations, is defined as

```
def pde_loss(self, pinn: PINN):
    x, y, t = self.environment.get_interior_points()
    ux,uy,p,duxdx,duxdy,duydx,duydy = pinn(x, y, t)
    duxdt = df(ux, t); d2uxdx = df(duxdx, x)
    d2uxdy = df(duxdy, y); dpdx = df(p,x)
```

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

```
duydt = df(uy, t); d2uydx = df(duydx, x)
d2uydy = df(duydy, y); dpdy = df(p, y)

loss1 = duxdt -d2uxdx - d2uxdy + dpdx
loss2 = duydt -d2uydx - d2uydy + dpdy
loss3 = duxdx + duydy
loss_duxdx = duxdx - df(ux, x)
loss_duxdy = duxdy - df(ux, y)
loss_duydx = duydx - df(uy, x)
loss_duydy = duydy - df(uy, y)

return loss1.pow(2).mean() + \
        loss2.pow(2).mean() + \
        loss3.pow(2).mean() + \
        loss_duxdx.pow(2).mean() + \
        loss_duxdy.pow(2).mean() + \
        loss_duydx.pow(2).mean() + \
        loss_duydy.pow(2).mean()
```

The second modification defines the boundary conditions. They are defined by using the hard constraints. We force u_x equal to zero on the entire boundary except the top boundary where it is equal to 1, u_y equal to zero on the entire boundary, and pressure p equal to zero at the middle point. The hard constraints in PINNs look as follows:

```
def force_up_stream(x, y):
    return torch.exp(-1000*(y-1)**2)

def zero_at_middle(x, y):
    return -torch.exp(-1000*(y-0.5)**2) \
        *torch.exp(-1000*(x-0.5)**2) + 1.0

def ux_constraint(logits, x, y):
    return logits * zero_dirichlet(x, y) \
    + force_up_stream(x, y)

def uy_constraint(logits, x, y):
    return logits * zero_dirichlet(x, y)

def p_constraint(logits, x, y):
    return logits * zero_at_middle(x, y)
```

Conclusions This paper explored the application of modern Physics-Informed Neural Networks (PINNs) to material science problems governed by linear elasticity and Navier-Stokes transient problems. For training of the linear elasticity non-stationary problem, we introduced randomized collocation point selection.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

For training of the Navier-Stokes time-dependent problem, we introduced the hard constraints for the initial and boundary conditions. The results demonstrated the potential of these methodologies to improve the performance and applicability of PINNs in modeling complex material science processes.

Main advantages of the PINN method compared to the classic simulations are the following. (1) Simplicity of the method. (2) Generic nature of neural networks - the approach is very similar for different problems and in many cases requires only a change in the loss function. There is no need to transform the equation to weak formulation and worry about choosing the right test and basis functions. (3) It can solve non-linear problems and inverse problems. (4) It can easily integrate measured data into training in addition to physics knowledge. (5) A single neural network can be trained for different sets of parameters.

However, there are certain disadvantages of PINNs. The solution is less accurate than the classical methods (like FEM); in more challenging problems to achieve proper convergence, certain "tricks" are required (like hard-constraint used in this paper). On top of that, there are issues related to all neural networks: choosing the right architecture and training parameters and difficulty in predicting what solution might help in certain problems. Overall, the topic of PINNs is definitely worth exploring, especially since the method is still relatively young. Neural networks are, after all, widely used in many various applications and perform extremely well. The main problem with PINNs, that is the accuracy of the solution, is improving fast, and it can even outperform classical simulators soon.

The future work following this paper may include: (1) Training generic PINN for different equation parameters. For example, linear elasticity equation has two so-called Lame coefficients (λ and μ), treated as constants in this work. It is possible to have them as additional input to the network. (2) Coupled multi-physics simulations including both elasticity and Navier-Stokes, or other challenging applications in material science. (3) Extension of the method to other classical problems solved by finite element method [5,6]. (4) Replacing PINNs by the Variational PINNs and including adaptive algorithms [24,27,26,25,8] for the test space.

References

1. DPM: A novel training method for physics-informed neural networks in extrapolation. The Thirty-Fifth AAAI Conference on Artificial Intelligence **35**. https://doi.org/10.1609/aaai.v35i9.16992, https://ojs.aaai.org/index.php/AAAI/article/view/16992
2. Alber, M., Tepole, A.B., Cannon, W.R., De, S., Dura-Bernal, S., Garikipati, K., Karniadakis, G., Lytton, W.W., Perdikaris, P., Petzold, L., Kuhl, E.: Integrating machine learning and multiscale modeling-perspectives, challenges, and opportunities in the biologica biomedical, and behavioral sciences. NPJ Digital Medicine **2** (2019). https://doi.org/10.1038/s41746-019-0193-y
3. Cai, S., Mao, Z., Wang, Z., Yin, M., Karniadakis, G.E.: Physics-informed neural networks (PINNs) for fluid mechanics: A review. Acta Mechanica Sinica **37**(12), 1727–1738 (2021)

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

4. Chen, Y., Lu, L., Karniadakis, G.E., Dal Negro, L.: Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Optics express **28**(8), 11618–11633 (2020)
5. Demkowicz, L.: Computing with *hp*-adaptive finite elements, vol. 1. Wiley (2006)
6. Demkowicz, L., Kurtz, J., Pardo, D., Paszynski, M., Rachowicz, W., Zdunek, A.: Computing with hp-ADAPTIVE FINITE ELEMENTS: Volume II Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications (1st ed.). Chapman and Hall/CRC (2007)
7. Geneva, N., Zabaras, N.: Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. Journal of Computational Physics **403** (2020). https://doi.org/10.1016/j.jcp.2019.109056
8. Goik, D., Jopek, K., Paszyński, M., Lenharth, A., Nguyen, D., Pingali, K.: Graph grammar based multi-thread multi-frontal direct solver with galois scheduler. Procedia Computer Science **29**, 960–969 (2014). https://doi.org/https://doi.org/10.1016/j.procs.2014.05.086, https://www.sciencedirect.com/science/article/pii/S1877050914002634, 2014 International Conference on Computational Science
9. Goswami, S., Anitescu, C., Chakraborty, S., Rabczuk, T.: Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. Theoretical and applied fracture machanics **106** (2020). https://doi.org/10.1016/j.tafmec.2019.102447
10. Guermond, J., Minev, P.: A new class of massively parallel direction splitting for the incompressible navier–stokes equations. Computer Methods in Applied Mechanics and Engineering **200**(23), 2083–2093 (2011). https://doi.org/https://doi.org/10.1016/j.cma.2011.02.007, https://www.sciencedirect.com/science/article/pii/S0045782511000429
11. Huang, X., Liu, H., Shi, B., Wang, Z., Yang, K., Li, Y., Wang, M., Chu, H., Zhou, J., Yu, F., Hua, B., Dong, B., Chen, L.: A universal PINNs method for solving Partial Differential Equations with a point source. Proceedings of the Fourteen International Joint Conference on Artificial Intelligence (IJCAI-22) pp. 3839–3846 (2022)
12. Jin, H., Mattheakis, M., Protopapas, P.: Physics-informed neural networks for quantum eigenvalue problems. In: 2022 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2022). https://doi.org/10.1109/IJCNN55064.2022.9891944
13. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: hp-VPINNs: Variational physics-informed neural networks with domain decomposition. Computer Methods in Applied Mechanics and Engineering **374**, 113547 (2021). https://doi.org/https://doi.org/10.1016/j.cma.2020.113547, https://www.sciencedirect.com/science/article/pii/S0045782520307325
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014), https://api.semanticscholar.org/CorpusID:6628106
15. Kissas, G., Yang, Y., Hwuang, E., Witschey, W.R., Detre, J.A., Perdikaris, P.: Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow MRI data using physics-informed neural networks. Computer Methods in Applied Mechanics and Engineering **358** (2020). https://doi.org/10.1016/j.cma.2019.112623
16. Ling, J., Kurzawski, A., Templeton, J.: Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. Journal of Fuild Mechanics **807**, 155–166 (2016). https://doi.org/10.1017/jfm.2016.615

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

17. Lu, L., Meng, X., Mao, Z., Karniadakis, G.E.: Deepxde: A deep learning library for solving differential equations. SIAM Review **63**(1), 208–228 (2021). https://doi.org/10.1137/19M1274067, https://doi.org/10.1137/19M1274067
18. Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., Johnson, S.G.: Physics-informed neural networks with hard constraints for inverse design. SIAM Journal on Scientific Computing **43**(6), B1105–B1132 (2021). https://doi.org/10.1137/21M1397908
19. Maczuga, P., Paszyński, M.: Influence of activation functions on the convergence of physics-informed neural networks for 1d wave equation. In: Mikyška, J., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M. (eds.) Computational Science – ICCS 2023. pp. 74–88. Springer Nature Switzerland, Cham (2023)
20. Mao, Z., Jagtap, A.D., Karniadakis, G.E.: Physics-informed neural networks for high-speed flows. Computer Methods in Applied Mechanics and Engineering **360**, 112789 (2020)
21. Marsden, J., Hughes, T.J.R.: Mathematical foundations of elasticity. Dover Publications, Inc. (1983)
22. Mishra, S., Molinaro, R.: Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. IMA Journal of Numerical Analysis **42**(2), 981–1022 (2022)
23. Nellikkath, R., Chatzivasileiadis, S.: Physics-informed neural networks for minimising worst-case violations in dc optimal power flow. In: 2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm). pp. 419–424 (2021). https://doi.org/10.1109/SmartGridComm51999.2021.9632308
24. Paszyńska, A., Paszyński, M., Grabska, E.: Graph transformations for modeling hp-adaptive finite element method with triangular elements. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) Computational Science – ICCS 2008. pp. 604–613. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
25. Paszyńska, A., Paszyński, M., Jopek, K., Woźniak, M., Goik, D., Gurgul, P., AbouEisha, H., Moshkov, M., Calo, V., Lenharth, A., Nguyen, D., Pingali, K.: Quasi-optimal elimination trees for 2d grids with singularities. Scientific Programming (1), 303024 (2015)
26. Paszyński, M., Grzeszczuk, R., Pardo, D., Demkowicz, L.: Deep learning driven self-adaptive hp finite element method. In: Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds.) Computational Science – ICCS 2021. pp. 114–121. Springer International Publishing, Cham (2021)
27. Paszyński, M., Paszyńska, A.: Graph transformations for modeling parallel hp-adaptive finite element method. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) Parallel Processing and Applied Mathematics. pp. 1313–1322. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
28. Peng, W., Zhang, J., Zhou, W., Zhao, X., Yao, W., Chen, X.: Idrlnet: A physics-informed neural network library (2021), https://arxiv.org/abs/2107.04320
29. Qin, S., Li, M., Xu, T., Dong, S.: Rar-pinn algorithm for the data-driven vector-soliton solutions and parameter discovery of coupled nonlinear equations. ArXiv **abs/2205.10230** (2022), https://api.semanticscholar.org/CorpusID:248965018
30. Raissi, M., Perdikaris, P., Karniadakis, G.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics **378**, 686–

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24

707 (2019). https://doi.org/https://doi.org/10.1016/j.jcp.2018.10.045, https://www.sciencedirect.com/science/article/pii/S0021999118307125
31. Rasht-Behesht, M., Huber, C., Shukla, K., Karniadakis, G.E.: Physics-informed neural networks (PINNs) for wave propagation and full waveform inversions. Journal of Geophysical Research: Solid Earth **127**(5), e2021JB023120 (2022)
32. Shin, Y., Darbon, J., Karniadakis, G.E.: On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. Communications in Computational Physics (2020), https://api.semanticscholar.org/CorpusID:225054225
33. Sun, F., Liu, Y., Sun, H.: Physics-informed spline learning for nonlinear dynamics discovery. Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21) pp. 2054–2061 (2021)
34. Sun, L., Gao, H., Pan, S., Wang, J.X.: Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. Computer Methods in Applied Mechanics and Engineering **361** (2020). https://doi.org/10.1016/j.cma.2019.112732
35. Taylor, C., Hughes, T.: Finite Element Programming of the Navier-Stokes Equations. Fluid mechanics its applications, Pineridge Press (1981), https://books.google.pl/books?id=wo0eAQAAIAAJ
36. Wandel, N., Weinmann, M., Neidlin, M., Klein, R.: Spline-PINN: Approaching PDEs without data using fast, physics-informed Hermite-spline CNNs. Proceedings of the AAAI Conference on Artificial Intelligence **36**(8), 8529–8538 (2022). https://doi.org/10.1609/aaai.v36i8.20830, https://ojs.aaai.org/index.php/AAAI/article/view/20830
37. Wang, S., Yu, X., Perdikaris, P.: When and why pinns fail to train: A neural tangent kernel perspective. Journal of Computational Physics **449**, 110768 (2022). https://doi.org/https://doi.org/10.1016/j.jcp.2021.110768, https://www.sciencedirect.com/science/article/pii/S002199912100663X
38. Yang, Y., Perdikaris, P.: Adversarial uncertainty quantification in physics-informed neural networks. Journal of Computational Physics **394**, 136–152 (2019). https://doi.org/10.1016/j.jcp.2019.05.027
39. Yuan, L., Ni, Y.Q., Deng, X.Y., Hao, S.: A-pinn: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations. Journal of Computational Physics **462**, 111260 (2022). https://doi.org/https://doi.org/10.1016/j.jcp.2022.111260, https://www.sciencedirect.com/science/article/pii/S0021999122003229
40. Łoś, M., Muga, I., Muñoz-Matute, J., Paszyński, M.: Isogeometric residual minimization (igrm) for non-stationary stokes and navier–stokes problems. Computers & Mathematics with Applications **95**, 200–214 (2021). https://doi.org/https://doi.org/10.1016/j.camwa.2020.11.013, https://www.sciencedirect.com/science/article/pii/S0898122120304417, recent Advances in Least-Squares and Discontinuous Petrov–Galerkin Finite Element Methods
41. Łoś, M., Paszyński, M.: Parallel shared-memory open-source code for simulations of transient problems using isogeometric analysis, implicit direction splitting and residual minimization (iga-ads-rm). Advances in Engineering Software **196**, 103723 (2024). https://doi.org/https://doi.org/10.1016/j.advengsoft.2024.103723, https://www.sciencedirect.com/science/article/pii/S0965997824001303

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_24