Physics Informed Neural Network Code for 2D Transient Problems (PINN-2DT) Compatible with Google Colab

Paweł Maczuga[1][0000−0002−5111−6981], Maciej Sikora[1][0009−0006−4465−2395], Tomasz Służalec[1][0000−0001−6217−4274], Marcin Szubert[1][0009−0005−3687−672X], Łukasz Sztangret[1][0000−0003−4872−406X], Danuta Szeliga[1][0000−0002−2915−8317], Marcin Łoś[1][0000−0002−8426−6345], Witold Dzwinel[1][0000−0001−8321−5928], Keshav Pingali[2], and Maciej Paszyński[1][0000−0001−7766−6052]

[1]AGH University of Krakow, Poland
maciej.paszynski@agh.edu.pl
[2] Oden Institute, The University of Texas at Austin, USA

Abstract. We present an open-source Physics Informed Neural Network environment for simulations of transient phenomena on two-dimensional rectangular domains, with the following features: (1) it is compatible with Google Colab which allows automatic execution on cloud environment; (2) it supports 2D linear or non-linear time-dependent PDEs; (3) it provides simple interface for definition of the residual loss, boundary condition and initial loss, together with their weights; (4) it support Neumann and Dirichlet boundary conditions; (5) it allows for customizing the number of layers and neurons per layer, as well as for arbitrary activation function; (6) the learning rate and number of epochs are available as parameters; (7) it automatically differentiates PINN with respect to spatial and temporal variables; (8) it provides routines for plotting the convergence (with running average), initial conditions learnt, 2D and 3D snapshots from the simulation and movies (9) it includes a library of problems: (a) non-stationary heat transfer; (b) atmospheric simulations including thermal inversion; (c) tumor growth simulations; and (d) the Stokes problem.

Keywords: Physics Informed Neural Networks, Colab, Atmospheric simulations, Tumor growth simulations, Material science simulations

1 Introduction

The goal of this paper is to replace the functionality of the time-dependent solver we published using isogeometric analysis and fast alternating directions solver [5–7] with the Physics Informed Neural Network (PINN) python library that can be easily executed on Colab. The PINN proposed in 2019 by Prof. Karniadakis revolutionized the way in which neural networks find solutions to initial-value problems described using partial differential equations [1] This method treats the neural network as a function approximating the solution of the given partial

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

differential equation $u(x) = PINN(x)$. After computing the necessary differential operators, the neural network and its appropriate differential operators are inserted into the partial differential equation. The residuum of the partial differential equation and the boundary-initial conditions are assumed as the loss function. The learning process involves sampling the loss function at different points by calculating the PDE residuum and the initial boundary conditions. The PINN methodology has had exponential growth in the number of papers and citations since its creation in 2019. It has multiple applications, from solid mechanics [15], geology [4], medical applications [11], and even the phase-field modeling of fracture [14]. Why use PINN solvers instead of classical or higher order finite element methods (e.g., isogeometric analysis) solvers? PINN/VPINN solvers have affordable computational costs. They can be easily implemented using pre-existing libraries and environments (like Pytorch and Google Colab). They are easily parallelizable, especially on GPU. They have great approximation capabilities, and they enable finding solutions to a family of problems. With the introduction of modern stochastic optimizers such as ADAM [3], they easily find high-quality minimizers of the loss functions employed.

In this paper, we present the PINN library with the following features

- It is implemented in Pythorch and compatible with Google Colab.
- It supports two-dimensional linear or non-linear problems defined on a rectangular domain.
- It is suitable for smooth problems without singularities resulting from large contrast material data.
- It enables the definition of the PDE residual loss function in the space-time domain.
- It supports the loss function for defining the initial condition.
- It provides loss functions for Neumann and Dirichlet boundary conditions.
- It allows for customization of the loss functions and their weights.
- It allows for defining an arbitrary number of layers of the neural network and an arbitrary number of neurons per layer.
- The learning rate, the kind of activation function, and a number of epochs are problem-specific parameters.
- It automatically performs differentiation of the PINN with respect to spatial and temporal variables.
- It provides tools for plotting the convergence of all the loss functions, together with the running average.
- It enables the plotting of the exact and learned initial conditions.
- It plots 2D or 3D snapshots from the simulations.
- It generates gifs with the simulation animation.

We illustrate our PINN-2DT code with four numerical examples. The first one concerns the model heat transfer problem. The second one is the simulation of the thermal inversion and the process of pollution removal by artificially generated shock waves, and the last one is the simulation of brain tumor growth.

There are the following available PINN libraries. First and most important is the DeepXDE library [12] by the team of Prof. Karniadakis. It is an extensive library with huge functionality, including ODEs, PDEs, complex geometries,

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

different initial and boundary conditions, and forward and inverse problems. It supports several tensor libraries such as TensorFlow, PyTorch, JAX, and PaddlePaddle. Another interesting library is IDRLnet [13]. It uses pytorch, numpy, and Matplotlib. This library is illustrated on four different examples, namely the wave equation, Allan-Cahn equations, Volterra integrodifferential equations, and variational minimization problems.

What is the novelty of our library? We do not claim to be better than these alternative high quality and multi-functionality libraries. The main point of our library is its simplicity of use and straight compatibility with Google Colab. It is a natural "copy" of the functionality of the IGA-ADS library [5] into the PINN methodology. It contains a simple, straightforward interface for solving different time-dependent problems. Our library can be executed without accessing the HPC center just by using the Google Colab.

The structure of the paper is the following. In Section 2, we recall the general idea of PINN on the example of the heat transfer problem. Section 3 is devoted to our code structure, from Colab implementation, model parameters, basic Python classes, how we define initial and boundary conditions, loss functions, how we run the training, and how we process the output. Section 4 provides four examples from heat transfer, wave equation, thermal inversion and the process of pollution removal by artificially generated shock waves, and tumor growth simulations. We conclude the paper in Section 5.

2 Physics Informed Neural Network for transient problems on the example of heat transfer problem

Let us consider a strong form of the exemplary transient PDE, the heat transfer problem. Find $u \in C^2(0,1)$ for $(x,y) \in \Omega = [0,1]^2$, $t \in [0,T]$ such that

$$\underbrace{\frac{\partial u(x,y,t)}{\partial t}}_{\text{time evolution}} \underbrace{-\epsilon\frac{\partial^2 u(x,y,t)}{\partial x^2} - \epsilon\frac{\partial^2 u(x,y,t)}{\partial y^2}}_{\text{diffusion term}} = \underbrace{f(x,y,t)}_{\text{forcing}}, (x,y,t) \in \Omega \times [0,T], \quad (1)$$

with initial condition $u(x,y,0) = u_0(x,y)$, and zero-Neumann boundary condition $\frac{\partial u}{\partial n} = 0$ $(x,y) \in \partial\Omega$. In the PINN approach, the neural network is the solution, namely

$$u(x,y,t) = PINN(x,y,t) = A_n\sigma\left(A_{n-1}\sigma(...\sigma(A_1[x,y,t] + B_1)...) + B_{n-1}\right) + B_n,$$

where A_i are matrices representing DNN layers, B_i represent bias vectors, and σ is the sigmoid, which as we have shown in [2], is the best choice for PINN. We define the loss function as the residual of the PDE

$$LOSS_{PDE}(x,y,t) = \left(\frac{\partial PINN(x,y,t)}{\partial t} - \epsilon\frac{\partial^2 PINN(x,y,t)}{\partial x^2} - \epsilon\frac{\partial^2 PINN(x,y,t)}{\partial y^2} - f(x,y,t)\right)^2. \quad (2)$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

We also define the initial condition loss $LOSS_{Init}(x,y,0) = (PINN(x,y,0) - u_0(x,y))^2$, as well as the loss of the residual of the boundary condition $LOSS_{BC}(x,y,t) = \left(\frac{\partial PINN(x,y,t)}{\partial n}(x,y,t) - 0\right)^2$.

3 Structure of the code

Our code is available at `https://github.com/pmaczuga/pinn-notebooks`

The code can be downloaded, openned in Google Colab, and executed in the fully automatic mode. There are the following model parameters to define

- `LENGTH`, `TOTAL_TIME`. The code works in the space-time domain, where the training is performed by selecting point along x, y and t axes. The `LENGTH` parameter defines the dimension of the domain along x and y axes. The domain dimension is `[0,LENGTH]x[0,LENGTH]x[0,TOTAL_TIME]`. The `TOTAL_TIME` parameter defines the length of the space-time domain along the t axis. It is the total time of the transient phenomena we want to simulate.
- `N_POINTS`. This parameter defines the number of points used for training. By default, the points are selected randomly along x, y, and t axes. It is easily possible to extend the code to support different numbers of points or different distributions of points along different axes of the coordinate system.
- `N_POINTS_PLOT`. This parameter defines the number of points used for probing the solution and plotting the output plots after the training.
- `WEIGHT_RESIDUAL`, `WEIGHT_INITIAL`, `WEIGHT_BOUNDARY`. These parameters define the weights for the training of residual, initial condition, and boundary condition loss functions.
- `LAYERS`, `NEURONS_PER_LAYER`. These parameters define the neural network by providing the number of layers and number of neurons per layer.
- `EPOCHS`, and `LEARNING_RATE` provide a number of epochs and the training rate for the training procedure.

Inside the `Loss` class, we provide interfaces for the definition of the loss functions. Namely, we define the `residual_loss`, `initial_loss` and `boundary_loss`. Since the initial and boundary loss is universal, and residual loss is problem specific, we provide fixed implementations for the initial and boundary losses, assuming that the initial state is prescribed in the `initial_condition` routine and that the boundary conditions are zero Neumann. The code can be easily extended to support different boundary conditions. We provide examples of loss functions in the next section.

We provide several routines for plotting the convergence of the loss function and for plotting the running average of the loss (see Fig. 1), for plotting the initial conditions in 2D and for plotting snapshots of the solution (see Fig. 2).

4 Examples of the instantiation

Heat transfer. We first present the instance of our library for the heat transfer problem described in Section 2. The residual loss function $LOSS_{PDE}(x,y,t) =$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

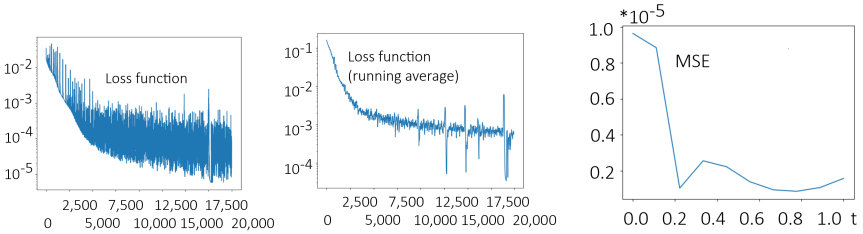

Fig. 1: Heat equation. Convergence of the residual loss function. Running average from the convergence of the residual loss function. Numerical error of the trained PINN solution to the heat transfer problem with manufactured solution.

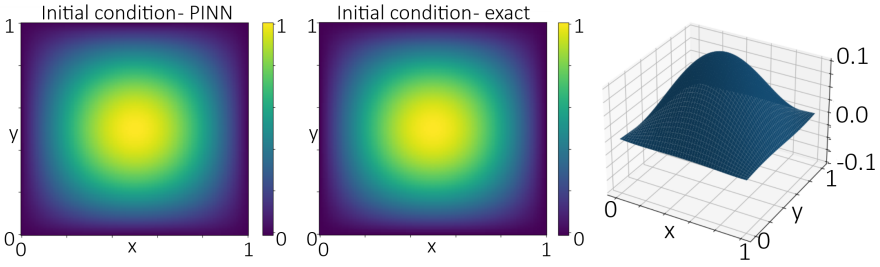

Fig. 2: Heat equation. Initial conditions in 2D. Snapshot from the simulation

$\left(\frac{\partial PINN(x,y,t)}{\partial t} - \frac{\partial^2 PINN(x,y,t)}{\partial x^2} - \frac{\partial^2 PINN(x,y,t)}{\partial y^2} - f(x,y,t)\right)^2$ translates into the following code

```
def residual_loss(self, pinn: PINN):
  x,y,t=get_interior_points(self.x_domain,
  self.y_domain,self.t_domain,self.n_points,pinn.device())
  u = f(pinn, x, y, t); z = self.floor(x, y)
  loss = dfdt(pinn, x, y, t, order=1) - \
      dfdx(pinn, x, y, t)**2-dfdy(pinn, x, y, t)**2
```

We employ the manufactured solution technique, where we assume the solution of the following form $u(x,y,t) = \exp^{-2\Pi^2 t} \sin \Pi x \sin \Pi y$, for $(x,y,t) \in [0,1]^2 \times [0,T]$. To obtain this particular solution, we set up the zero Dirichlet boundary conditions, which require the following code

```
def boundary_loss_dirichlet(self, pinn: PINN):
  down,up,left,right=get_boundary_points(self.x_domain,
  self.y_domain,self.t_domain,self.n_points,pinn.device())
  x_down,  y_down,  t_down    = down
  x_up,    y_up,    t_up      = up
  x_left,  y_left,  t_left    = left
  x_right, y_right, t_right   = right
  loss_down  = f( pinn, x_down,  y_down,  t_down  )
  loss_up    = f( pinn, x_up,    y_up,    t_up    )
```

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

```
  loss_left  = f( pinn, x_left,  y_left,  t_left  )
  loss_right = f( pinn, x_right, y_right, t_right )
  return loss_down.pow(2).mean()+loss_up.pow(2).mean()+ \
    loss_left.pow(2).mean()+loss_right.pow(2).mean()
```

We also setup the initial state $u_0(x, y) = \sin(\Pi x)\sin(\Pi y)$ which translates into the following code

```
def initial_condition(x:torch.Tensor,y:torch.Tensor)->
    torch.Tensor:
    res = torch.sin(torch.pi*x) * torch.sin(torch.pi*y)
    return res
```

The default setup of the parameters for this simulation is the following:

```
LENGTH = 1.              TOTAL_TIME = 1.
N_POINTS = 15            N_POINTS_PLOT = 150
WEIGHT_RESIDUAL = 1.0    WEIGHT_INITIAL = 1.0
WEIGHT_BOUNDARY = 1.0    LAYERS = 4
NEURONS_PER_LAYER = 80   EPOCHS = 20_000
LEARNING_RATE = 0.002
```

The convergence of the loss function and the running average of the loss are presented in Fig. 1. The comparison of exact and trained initial conditions and the snapshot from the simulation is presented in Fig. 2 for time moment $t = 0.1$. The mean square error of the computed simulation is presented in Fig. 1. We can see the high accuracy of the trained PINN results.

Thermal inversion and the process of pollution removal by artificially generated shock waves. In this example, we aim to model the thermal inversion effect. The numerical results presented in this section are the PINN version of the thermal inversion simulation performed using isogeometric finite element method code [5] described in [9]. The scalar field u in our simulation represents the water vapor forming a cloud. The source represents the evaporation of the cloud evaporation of water particles near the ground. The thermal inversion effect is obtained by introducing the advection field as the gradient of the temperature. Following [10] we define $\frac{\partial T}{\partial y} = -2$ for lower half of the domain ($y < 0.5$), and $\frac{\partial T}{\partial y} = 2$ for upper half of the domain ($y > 0.5$).

We focus on advection-diffusion equations in the strong form. We seek the cloud vapor concentration field $[0,1]^2 \times [0,1] \ni (x, y, t) \to u(x, y, t) \in \mathcal{R}$

$$\begin{aligned} \frac{\partial u(x,y,t)}{\partial t} + b(x,y,t)\cdot\nabla u(x,y,t) - \nabla\cdot(K(x,y)\ \nabla u(x,y,t)) & \\ = f(x,y,t), \ (x,y,t) \in \Omega \times (0,T], & \\ \nabla u \cdot n = 0, \qquad (x,y,t) \in \partial\Omega \times (0,T], & \\ u(x,y,0) = u_0(x,y), \ (x,y,t) \in \Omega \times 0. & \end{aligned} \tag{3}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

This PDE translates into

$$\begin{aligned} \frac{\partial u(x,y,t)}{\partial t} + \frac{\partial T(y)}{\partial y}\frac{\partial u(x,y,t)}{\partial y} - 0.1\frac{\partial u(x,y,t)}{\partial x^2} - 0.01\frac{\partial u(x,y,t)}{\partial y^2} \\ = f(x,y,t),\ (x,y,t) \in \Omega \times (0,T], \\ \nabla u \cdot n = 0, \qquad (x,y,t) \in \partial\Omega \times (0,T], \\ u(x,y,0) = u_0(x,y), (x,y,t) \in \Omega \times 0. \end{aligned} \tag{4}$$

We define the loss function as the residual of the PDE

$$\begin{aligned} LOSS_{PDE}(x,y,t) = \Bigg(\frac{\partial PINN(x,y,t)}{\partial t} + \frac{\partial T(y)}{\partial y}\frac{\partial PINN(x,y,t)}{\partial y} \\ - 0.1\frac{\partial PINN(x,y,t)}{\partial x^2} - 0.01\frac{\partial PINN(x,y,t)}{\partial y^2} - f(x,y,t) \Bigg)^2 . \end{aligned} \tag{5}$$

Followng [9] we modify the loss function to introduce the term modeling the artificially generated shock wave. The convergence of the loss function is summarized in Fig. 3. The snapshots from the simulations are presented in Fig. 4. In the thermal inversion, the cloud vapor that evaporated from the ground stays close to the ground, due to the distribution of the temperature gradients. To

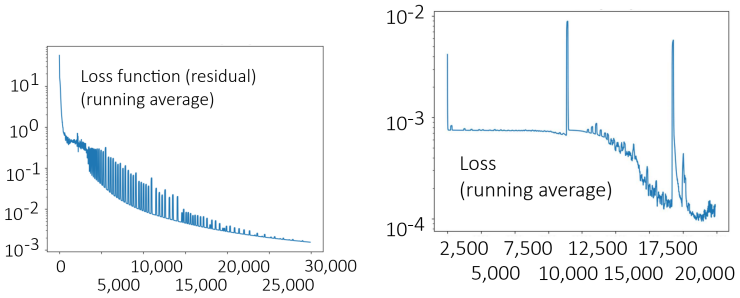

Fig. 3: Thermal inversion: Convergence of the loss function. Tumor growth: Convergence of the loss function.

model the generation of the artificial shock waves, following the idea described in [9], we modify the advection by adding the term responsible for the generated shock wave.

Tumor growth. The next example concerns the brain tumor growth, as described in [11]. We seek the tumor cell density $[0,1]^2 \times [0,1] \ni (x,y,t) \to u(x,y,t) \in \mathcal{R}$, such that

$$\begin{aligned} \frac{\partial u(x,y,t)}{\partial t} = \nabla \cdot (D(x,y)\nabla u(x,y,t)) + \rho u(x,y,t)\,(1 - u(x,y,t)) \\ (x,y,t) \in \Omega \times (0,T], \\ \nabla u \cdot n = 0,\ (x,y,t) \in \partial\Omega \times (0,T], \quad u(x,y,0) = u_0(x,y),\ (x,y,t) \in \Omega \times 0, \end{aligned} \tag{6}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

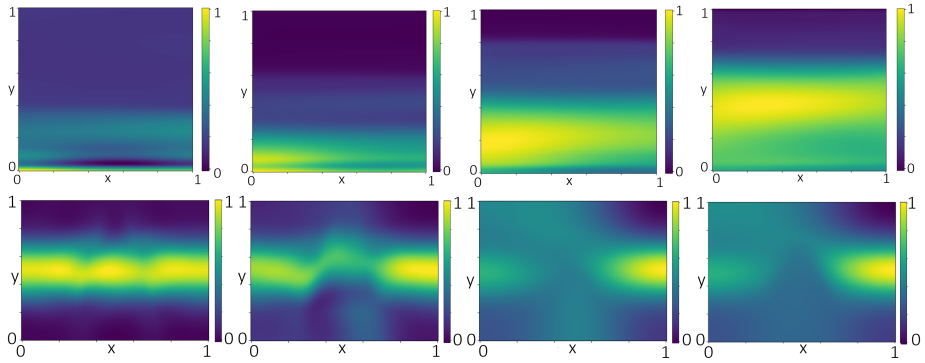

Fig. 4: **Top panel**: Thermal inversion. **Bottom panel**: Hail cannon simulation.

which translates into

$$\frac{\partial u(x,y,t)}{\partial t} - \frac{\partial D(x,y)}{\partial x}\frac{\partial u(x,y,t)}{\partial x} - D(x,y)\frac{\partial^2 u(x,y,t)}{\partial x^2}$$
$$-\frac{\partial D(x,y)}{\partial y}\frac{\partial u(x,y,t)}{\partial y} - D(x,y)\frac{\partial^2 u(x,y,t)}{\partial x^y} - \rho u(x,y,t)\left(1-u(x,y,t)\right) = 0.$$

Here, $D(x,y)$ represents the tissue density coefficient, where $D(x,y) = 0.13$ for the white matter, $D(x,y) = .013$ for the gray matter, and $D(x,y) = 0$ for the cerebrospinal fluid (see [11] for more details). Additionally, $\rho = 0.025$ denotes the proliferation rate of the tumor cells. We simplify the model

$$\frac{\partial u(x,y,t)}{\partial t} - D(x,y)\frac{\partial^2 u(x,y,t)}{\partial x^2} - D(x,y)\frac{\partial^2 u(x,y,t)}{\partial x^y} - \rho u(x,y,t)\left(1-u(x,y,t)\right) = 0. \tag{7}$$

We define the loss function as the residual of the PDE

$$LOSS_{PDE}(x,y,t) + \left(\frac{\partial u(x,y,t)}{\partial t} - \frac{\partial D(x,y)}{\partial x}\frac{\partial u(x,y,t)}{\partial x} - D(x,y)\frac{\partial^2 u(x,y,t)}{\partial x^2}\right.$$
$$\left. -\frac{\partial D(x,y)}{\partial y}\frac{\partial u(x,y,t)}{\partial y} - D(x,y)\frac{\partial^2 u(x,y,t)}{\partial x^y} - \rho u(x,y,t)\left(1-u(x,y,t)\right)\right)^2 \tag{8}$$

We summarize in Fig. 3 the convergence of the loss function. Additionally, Fig. 5 presents the snapshots from the simulation.

Stationary non-linear Navier-Stokes problem. Let us focus on the stationary cavity flow problem [16] described with the stationary non-linear Navier-Stokes equation for the incompressible fluid; see Figure 7. 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

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

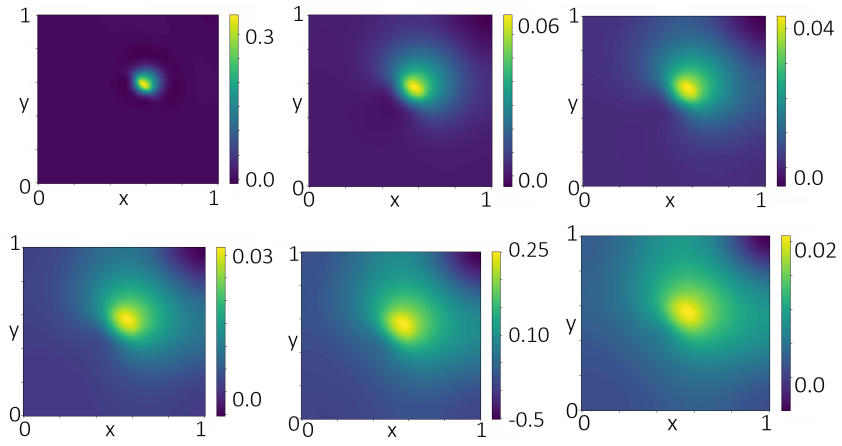

Fig. 5: Tumor growth. Snapshots from the simulation.

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. Let $\Omega = (0,1)^2$ be the open boundary. The cavity flow problem reads: Find velocity u and pressure field p such that:

$$\begin{cases} -\dfrac{\Delta \mathbf{u}}{Re} + u \cdot \nabla u + \nabla p = 0 & \text{in } \Omega, \\ \nabla \cdot \mathbf{u} = 0 & \text{in } \Omega, \\ \mathbf{u} = h & \text{in } \Gamma \end{cases}, \; h = \begin{cases} 0 & x \in (0,1),\ y = 0 \\ 0 & x \in \{0,1\},\ y \in (0, 1-\epsilon) \\ 1 & x \in (0,1),\ y = 1 \\ \dfrac{\epsilon - y + 1}{\epsilon} & x \in \{0,1\},\ y \in (1-\epsilon, 1) \end{cases} \tag{9}$$

System (9) can be rewritten as

$$w_1(x_1,x_2) = \frac{\partial u_1(x_1,x_2)}{\partial x_1},\ w_2(x_1,x_2) = \frac{\partial u_1(x_1,x_2)}{\partial x_2}$$
$$z_1(x_1,x_2) = \frac{\partial u_2(x_1,x_2)}{\partial x_1},\ z_2(x_1,x_2) = \frac{\partial u_2(x_1,x_2)}{\partial x_2}$$

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

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

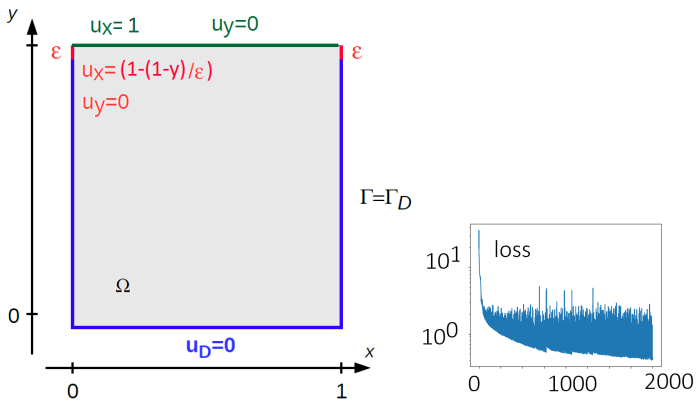

Fig. 6: Non-stationary cavity flow problem. Convergence of PINN training.

Despite the non-linear problem, we easily define the following residuals

$$RES_0 = \frac{\partial u_1}{\partial x_1} - w_1, \quad RES_1 = \frac{\partial u_1}{\partial x_2} - w_2, \quad RES_2 = \frac{\partial u_2}{\partial x_1} - z_1,$$
$$RES_3 = \frac{\partial u_2}{\partial x_2} - z_2, \quad RES_4 = -\frac{\partial w_1}{\partial x_1} - \frac{\partial w_2}{\partial x_2} + \frac{\partial p}{\partial x_1} + u_1 w_1 + u_2 w_2 - f_1,$$
$$RES_5 = -\frac{\partial z_1}{\partial x_1} - \frac{\partial z_2}{\partial x_2} + \frac{\partial p}{\partial x_2} + u_1 z_1 + u_2 z_2 - f_2,$$
$$RES_6 = \frac{\partial u_1(x_1, x_2)}{\partial x_1} + \frac{\partial u_2(x_1, x_2)}{\partial x_2}.$$

The Stokes code is available at `https://colab.research.google.com/drive/15eDVY7DyRfu5ugJRISETPjk-A-W6wA88`

The obtained solution is shown in Figure 7. The Colab execution time on L4 graphic card, for 20,000 iterations with 4 layers of 200 neurones (a total of $200 \times 200 \times 3 = 24,000,000$ parameters), with 100×100 integration points and learning rate 0.005 is around 15 minutes. The benefit of PINN solution with respect to traditional finite element method solution is that it does not require any linearization or special stabilization methods. The non-linear PDEs are (split into a first order system though) are directly implemented in the loss function.

5 Conclusions

We have created a code `https://github.com/pmaczuga/pinn-notebooks` that can be downloaded and opened in the Google Colab. It can be automatically executed using Colab functionality. The code provides a simple interface for running two-dimensional time-dependent simulations on a rectangular grid. It provides an interface to define residual loss, initial condition loss, and boundary condition

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

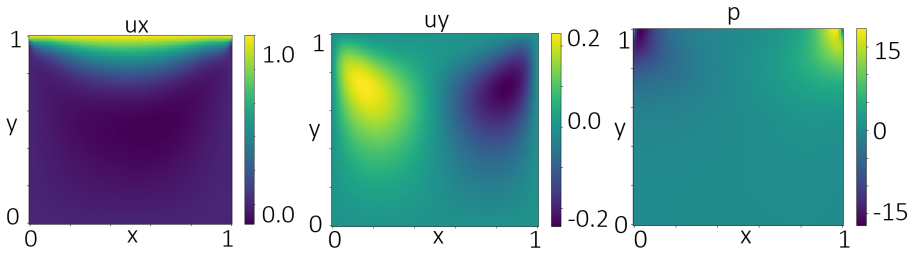

Fig. 7: The first and second components of the velocity vector field for the cavity flow problem. The pressure scalar field for the cavity flow problem.

loss. It provides examples of Dirichlet and Neumann boundary conditions. The code also provides routines for plotting the convergence, generating snapshots of the simulations, verifying the initial condition, and generating the animated gifs. We also provide four examples, the heat transfer, the thermal inversion from advection-diffusion equations, the brain tumor model, and the Stokes problem. The future work may involve development of the Variational PINN library with adaptive test space, following [17–22]. The PINN methods are naturally slower than finite element method codes, but they enable easy approximation of non-linear PDEs, without the necessity of linearization of the formulation. The non-linear problem can be directly incorporated into the residual.

Acknowledgements This work was supported by the program "Excellence initiative - research university" for the AGH University of Science and Technology. The visit of Maciej Paszyński at Oden Institute was partially supported by J. T. Oden Research Faculty Fellowship.

A Code sniplets

Thermal inversion and shock wave generation The residual loss includes now the vertical temperature gradient `dTy`, the diffusion variables `Kx` and `Ky`, and the source term `source`.

```
def residual_loss(self, pinn: PINN):
     x, y, t = get_interior_points(self.x_domain, self.
         y_domain, self.t_domain,
     self.n_points, pinn.device())
     loss = dfdt(pinn, x, y, t).to(device)
     - self.dTy(y, t)*dfdy(pinn, x, y, t).to(device)
     - self.Kx*dfdx(pinn, x, y, t,order=2).to(device)
     - self.Ky*dfdy(pinn, x, y, t, order=2).to(device)
     - self.source(y,t).to(device)
     return loss.pow(2).mean
def source(self,y,t):
     d=0.7; res=torch.clamp((torch.cos(t*math.pi)-d)*1/(1-
         d), min=0)
```

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

```
        res2 = (150 - 1200 * y) * res
        res3 = torch.where(t <= 0.3, res2, 0)
        res4 = torch.where(y <= 0.125, res3, 0)
        return res4.to(device)
```

During the training, we use the following global parameters

```
LENGTH = 1.                 WEIGHT_BOUNDARY = 10.0
TOTAL_TIME = 1.             LAYERS = 2
N_POINTS = 15               NEURONS_PER_LAYER = 600
N_POINTS_PLOT = 150         EPOCHS = 30_000
WEIGHT_RESIDUAL = 20.0      LEARNING_RATE = 0.002
WEIGHT_INITIAL = 1.0
```

The shock wave implementation in the advection term involves the following modifications to `dTy` routine:

```
def dTy(self,x:torch.Tensor,y:torch.Tensor,t:torch.Tensor)
    ->torch.Tensor:
  sin_term=torch.sin(torch.pi*t/2)*torch.sin(torch.pi*x)
      -0.8*torch.sin(torch.pi*t/2)
  es_threshold=torch.maximum(5*sin_term,torch.tensor(0.0,
      device=x.device))
  mask=(y<res_threshold);peak_value=torch.tensor(-3.0,
      device=device,dtype=x.dtype)
  def_val=torch.tensor(0.0,device=device,dtype=x.dtype)
  foff_rate=torch.tensor(1.0,device=device,dtype=x.dtype)
  inverted_mask_np=(~mask).cpu().numpy()
  distances_np=scipy.ndimage.distance_transform_edt(
      inverted_mask_np)
  distances=torch.from_numpy(distances_np).to(device=
      device,dtype=x.dtype)
  falloff_values=torch.clamp(peak_value+foff_rate*
      distances,max=def_val)
  final_grid=torch.where(mask.to(device),peak_value.to(
      device),falloff_values.to(device))
  return final_grid.to(device)*(-2)
```

Tumor growth simulations The loss function involves now the variable diffusion terms and the proliferation terms

```
def residual_loss(self, pinn: PINN):
    x, y, t = get_interior_points(
       self.x_domain, self.y_domain, self.t_domain,
       self.n_points, pinn.device())
    rho = 0.025
    def D_fun(x, y) -> torch.Tensor:
        res = torch.zeros(x.shape, dtype=x.dtype, device=
           pinn.device())
        dist = (x-0.5)**2 + (y-0.5)**2
        res[dist < 0.25] = 0.13; res[dist < 0.02] = 0.013
        return res
```

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

```
    D = D_fun(x, y); u = f(pinn, x, y, t)
    loss = dfdt(pinn,x,y,t)-D*dfdx(pinn,x,y,t,order=2) \
            - D*dfdy(pinn,x,y,t,order=2)-rho*u*(1-u)
    return loss.pow(2).mean()
```

The initial and boundary condition loss functions are unchanged. The initial state is given as follows:

```
def initial_condition(x: torch.Tensor, y: torch.Tensor) ->
    torch.Tensor:
 d = torch.sqrt((x-0.6)**2 + (y-0.6)**2)
 res = -d**2 - 4*d + 0.4; res = res * (res > 0)
 return res
```

We setup the following model parameters

```
LENGTH = 1.              WEIGHT_BOUNDARY = 1.0
TOTAL_TIME = 1.          LAYERS = 4
N_POINTS = 20            NEURONS_PER_LAYER = 80
N_POINTS_PLOT = 150      EPOCHS = 50_000
WEIGHT_RESIDUAL = 1.0    LEARNING_RATE = 0.005
WEIGHT_INITIAL = 1.0
```

Non-linear stationary Navier-Stokes The residual loss for non-linear stationary Navier-Stokes is the following

```
def calculate_loss(pinns: list[PINN], x: torch.Tensor, y:
    torch.Tensor) -> torch.Tensor:
  ux = pinns[0]; uy = pinns[1]; p = pinns[2]
  duxdx = pinns[3]; duxdy = pinns[4]; duydx = pinns[5]
  duydy = pinns[6]; u1 =  f(ux, x, y); u2 =  f(uy, x, y)
  du1dx = f(duxdx, x, y); du1dy = f(duxdy, x, y)
  du2dx = f(duydx, x, y); du2dy = f(duydy, x, y)
  uxdotgradux = u1 * du1dx + u2 * du1dy
  uydotgraduy = u1 * du2dx + u2 * du2dy
  d2uxdx = dfdx(duxdx,x,y); d2uxdy = dfdy(duxdy,x,y)
  dpdx = dfdx(p,x,y,order=1); d2uydx = dfdx(duydx, x, y)
  d2uydy = dfdy(duydy, x, y); dpdy = dfdy(p,x,y)
  loss1 = -d2uxdx - d2uxdy + dpdx + uxdotgradux
  loss2 = -d2uydx - d2uydy + dpdy + uydotgraduy
  loss3 = duxdx(x, y) + duydy(x, y)
  loss_duxdx = duxdx(x, y) - dfdx(ux, x, y)
  loss_duxdy = duxdy(x, y) - dfdy(ux, x, y)
  loss_duydx = duydx(x, y) - dfdx(uy, x, y)
  loss_duydy = duydy(x, y) - dfdy(uy, x, 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 boundary conditions and zero pressure value at the center point are enforced using the hard constraints.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

```
def zero_dirich(x,y):return 20*x*(1-x)*y*(1-y)
def force_up_stream(x,y):return torch.exp(-1000*(y-1)**2)
def zero_middle(x,y):return -torch.exp(-1000*(y-0.5)**2)*
    torch.exp(-1000*(x-0.5)**2)+1.0
def ux_constraint(logs,x,y):return logs*zero_dirich(x,y)
     +force_up_stream(x,y)
def uy_constraint(logs,x,y):return logs*zero_dirich(x,y)
def p_constraint(logs,x,y):return logs*zero_middle(x,y)
```

The code requires an extension to support PINN with vector output

```
ux_pinn = PINN(LAYERS, NEURONS_PER_LAYER, hard_constraint=
    ux_constraint)
uy_pinn = PINN(LAYERS, NEURONS_PER_LAYER, middle_layers=
    ux_pinn.middle_layers, hard_constraint=uy_constraint)
p_pinn  = PINN(LAYERS, NEURONS_PER_LAYER, middle_layers=
    ux_pinn.middle_layers, hard_constraint=p_constraint)
duxdx_pinn  = PINN(LAYERS, NEURONS_PER_LAYER,
    middle_layers=ux_pinn.middle_layers)
duxdy_pinn  = PINN(LAYERS, NEURONS_PER_LAYER,
    middle_layers=ux_pinn.middle_layers)
duydx_pinn  = PINN(LAYERS, NEURONS_PER_LAYER,
    middle_layers=ux_pinn.middle_layers)
duydy_pinn  = PINN(LAYERS, NEURONS_PER_LAYER,
    middle_layers=ux_pinn.middle_layers)
pinns = [ux_pinn, uy_pinn, p_pinn, duxdx_pinn, duxdy_pinn,
     duydx_pinn, duydy_pinn]
multiPINN = MultiPINN(LAYERS, NEURONS_PER_LAYER, pinns=
    pinns).to(DEVICE)
```

References

1. M. Raissi, P.Perdikaris, G.E.Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, 378 (2019) 686-707.
2. P. Maczuga, M. Paszyński, Influence of Activation Functions on the Convergence of Physics-Informed Neural Networks for 1D Wave Equation, Computational Science – ICCS 2023: 23rd International Conference, Prague, Czech Republic, July 3-5, 2023, Proceedings, Part I, (2023) 74–88.
3. D. P. Kingma, J. Lei Ba, ADAM: A method for stochastic optimization, arXiv:1412.6980 (2014)
4. Y. Chen, Y. Xu, L. Wang, T. Li, Modeling water flow in unsaturated soils through physics-informed neural network with principled loss function, Computers and Geotechnics, 161 (2023) 105546
5. M. Łoś, M. Woźniak, M. Paszyński, A. Lenharth, K. Pingali, IGA-ADS : Isogeometric Analysis FEM using ADS solver, Computer & Physics Communications, 217 (2017) 99-116.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13

6. M. Łoś, M. Paszyński, A. Kłusek, Witold Dzwinel, Application of fast isogeometric L2 projection solver for tumor growth simulations, Computer Methods in Applied Mechanics and Engineering, 316 (2017) 1257-1269.
7. M. Łoś, A. Kłusek, M. Amber Hassaan, K. Pingali, W. Dzwinel, M. Paszyński. Parallel fast isogeometric L2 projection solver with GALOIS system for 3D tumor growth simulations, Computer Methods in Applied Mechanics and Engineering, 343 (2019) 1-22.
8. P. Maczuga, A. Oliver-Serra, A. Paszyńska, E. Valseth, M. Paszyński, Graph-grammar based algorithm for asteroid tsunami simulations, Journal of Computational Science, 64 (2022) 101856
9. K. Misan, W. Ormaniec, A. Kania, M. Kozieja, M. Łoś, D. Gryboś, J. Leszczyński, M. Paszyński, Fast isogeometric analysis simulationsof a process of air pollution removal byartificially generated shock waves, Computational Science – ICCS 2022: 22rd International Conference, London, UK, June 21-23, 2022, Proceedings, Part I, (2022) 298-311
10. U.S. Standard Atmosphere vs. Altitude, Engineering ToolBox (2003) https://www.engineeringtoolbox.com/standard-atmosphere-d_604.html,
11. A. Zhu, J. Vo, J. Lowengrub, Accelerating Parameter Inference in Diffusion-Reaction Models of Glioblastoma Using Physics-Informed Neural Networks, SIAM Undergraduate Research Online (2022)
12. L. Lu, X. Meng, Z. Mao, G. Em Karniadakis, DeepXDE: A deep learning library for solving differential equations, SIAM Review, 63(1), (2021) 208-228.
13. W. Peng, J. Zhang, W. Zhou, X. Zhao, W. Yao, X. Chen, IDRLnet: A Physics-Informed Neural Network Library, arxiv.2107.04320 (2021)
14. S. Goswami, C. Anitescu, S. Chakraborty, T. Rabczuk, Transfer learning enhanced physics informed neural network for phase-field modeling of fracture, Theoretical and Applied Fracture Mechanics, 106 (2020) 102447
15. E. Haghighat, M. Raissi, A. Moure, H. Gomez, R. Juanes, A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics, Computer Methods in Applied Mechanics and Engineering, 379 (2021) 1879-2138,
16. P. Matuszyk, M. Paszyński, Fully automatic hp adaptive finite element method for the Stokes problem in two dimensions, Computer Methods in Applied Mechanics and Engineering, 197(51–52) (2008) 4549-4558.
17. L. Demkowicz, Computing with hp-adaptive finite elements, Vol. 1, Wiley (2006).
18. L. Demkowicz, J. Kurtz, D. Pardo, M. Paszynski, W. Rachowicz, A. Zdunek, Computing with hp-ADAPTIVE FINITE ELEMENTS: Volume II Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications (1st ed.), Chapman and Hall/CRC (2007).
19. A. Paszyńska, M. Paszyński, E. Grabska, Graph Transformations for Modeling hp-Adaptive Finite Element Method with Triangular Elements Lecture Notes in Computer Science, 5103 (2008) 604-613.
20. D. Goik, K. Jopek, M. Paszyński, A. Lenharth, D. Nguyen, K. Pingali, Graph grammar based multi-thread multi-frontal direct solver with Galois scheduler Procedia Computer Science 29 (2014) 960-969.
21. M. Paszyński, A. Paszyńska, Graph Transformations for Modeling Parallel *hp*-Adaptive Finite Element Method, Parallel Processing and Applied Mathematics: 7th International Conference, (2007) 1313-1322.
22. A. Paszyńska, M.Paszyński, K. Jopek, M. Woźniak, D. Goik, P. Gurgul, H. AbouEisha, M. Moshkov, V. M. Calo, A. Lenharth, D. Nguyen, K. Pingali, Quasi-Optimal Elimination Trees for 2D Grids with Singularities, Scientific Programming, 1 (2015) 303024.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_13