Study the behavior of a fixed bed catalytic reactor with PINNs method

Dinh-Nam Mai[1], Phuc-Luan Nguyen[1], Doan-Kien Thai[1]

[1]Students at National Institute of Applied Sciences of Toulouse, France

Advisor: Marie-Jose Huguet [2] ,Liantsoa Onditi-Randriambololona[2],Arnaud Cockx[2],Philippe Schmitz[2]
[2]National Institute of Applied Sciences of Toulouse, France

ABSTRACT

In this study, we employ Physics-Informed Neural Networks (PINNs) to solve the Navier-Stokes equations and the heat balance equation within a 2D stationary model of fluid flow between two parallel plates. Traditional computational approaches for solving partial differential equations PDEs, such as Computational Fluid Dynamics CFD simulations (Appendix A), can be quite costly. CFD simulations numerically solve the PDEs governing fluid flow and heat transfer by discretizing the computational domain and iteratively solving the equations, often requiring significant computational resources. PINNs offer a novel approach by integrating PDEs and boundary conditions into the neural network's loss function, guiding the model to learn from the governing physics of the problem. The goal of this research is to accurately predict temperature, pressure, and velocity distributions within the fluid flow model and compare with the results from the CFD solver (Appendix A) . By minimizing the loss function, the trained neural network model is expected to provide reliable solutions that satisfy the PDE equations and boundary conditions. Through this approach, we aim to demonstrate the potential of PINNs in streamlining and enhancing the computational efficiency of solving fluid dynamics and heat transfer problems.

Keywords: Physics-Informed Neural Networks (PINNs), Navier-Stokes equations, heat balance equation, 2D stationary model, incompressible flow, CFD solver.

1 Introduction

Physics-Informed Neural Networks (PINNs) have presented a significant initiative in solving Partial Differential Equations (PDEs), which directly integrate deep learning with physical laws, enabling the incorporation of known physical information into the architecture of neural networks [1]. This integration not only enhances the accuracy and efficiency of solving PDEs but also aids in the discovery of new physical laws from data [2],in this study, we focus on the first aspect: solving PDEs and describing their results. Recent advancements have introduced variations like the Variational Physics-Informed Neural Networks (VPINNs), which employ a Petrov-Galerkin approach to lower the training costs and increase solution accuracy by using neural networks as trial spaces and orthogonal polynomials as test spaces [3]. Additionally, the development of distributed and self-training PINNs has addressed scalability and accuracy issues, making these networks more effective in solving time-dependent and large domain PDEs [4]. This expansion in applicability signifies the potential of PINNs as a tool for other computational physics' problems.

The modeling of fixed-bed catalytic reactors presents several challenges, particularly in accurately representing the behavior and interactions of physical and chemical processes such as thermal phenomena and flow dynamics. Current research indicates a notable gap in the integration of the Navier-Stokes equations and heat balance equations for developing an optimal model of these reactors [5]. The Navier-Stokes equations, which describe fluid dynamics, are not straightforward to solve in the context of fixed-bed reactors due to the complex interplay between flow and the packed bed structure. Traditional methods like the Computational Fluid Dynamics (CFD) (Appendix A) are used, but they often require simplifications that may not capture all the necessary dynamics of fluid flow and heat transfer effectively [6]. We can also apply Artificial feed-forward neural network (AFFNN), a type of artificial neural network where connections between the nodes do not form cycles; the information moves in one direction—from input nodes, through hidden nodes (if any), to output nodes. Research has shown that fixed-bed reactors could be well-simulated with trained AFFNN , though adaptive linear kinetic rate Thiele Modulus method gave more desirable precision [7].

This study aims to accurately simulate the behavior of a fixed-bed catalytic reactor using the PINNs method, which is anticipated to enhance the precision of model [8]. We integrates the Navier-Stokes equations for fluid dynamics and heat equations for thermal phenomena with neural network to enhance the predictive accuracy of computational models under the constraints of known physics. Furthermore, we will carefully optimize the weights and biases of the models using different optimization approaches, then supervise our model by comparison between the optimizations' methods. This ensures that predictions adhere strictly to fundamental scientific laws. The entire modeling process and its validation are implemented using DeepXDE [9] uses TensorFlow as the default backend to perform neural network related calculations, providing a framework for demonstrating the practical application and effectiveness of our proposed model. This approach suggests the feasibility of our model an into complex chemical engineering systems.

2 2D Navier-Stokes Equations and heat balance equation

In this section, we discuss the governing equations for incompressible fluid flow in two dimensions (2D), known as the Navier-Stokes equations. These equations describe the motion of fluid substances such as liquids and gases. The primary variables in these equations are the fluid velocity $\mathbf{u} = (u, v)$, pressure p, and temperature T.

2.1 Assumptions and Characteristics of the system

The assumptions and characteristics of the system under consideration include:

- The flow is incompressible and steady-state, meaning the fluid density is constant and the equations do not depend on time.
- The fluid has constant properties, such as density ρ, dynamic viscosity μ, specific heat capacity c_p, and thermal conductivity k.
- The system is not subject to any external forces or influences, including gravitational forces.
- The characteristic length $2D$ and characteristic velocity U are chosen based on the problem geometry and flow conditions.

2.2 Dimensional Navier-Stokes Equations

The dimensional Navier-Stokes equations for incompressible flow [10] are given by :

Continuity Equation:

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

In Cartesian coordinates:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{2}$$

Momentum Equations:

$$\rho(\mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla p + \mu \nabla^2 \mathbf{u} \tag{3}$$

In Cartesian coordinates, the momentum equations can be expressed as:
X-direction:

$$\rho \left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} \right) = -\frac{\partial p}{\partial x} + \mu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{4}$$

Y-direction:

$$\rho \left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} \right) = -\frac{\partial p}{\partial y} + \mu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \tag{5}$$

Energy Equation:

$$\rho c_p \left(u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} \right) = k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) \tag{6}$$

In Cartesian coordinates:

$$\rho c_p \left(u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} \right) = k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) \tag{7}$$

2.3 Dimensionless Navier-Stokes Equations

We used dimensionless forms of the Navier-Stokes equations for our problem because PINNs method involves substituting estimates into the equations to calculate the total error of the system, which is then minimized. Normalizing the equations ensures that all variables have values within the range [0, 1], which leads to a more accurate minimization process. Without normalization, the optimization could be skewed toward minimizing the variables with the largest scales, leading to less accurate results for other variables. We introduce the following dimensionless variables:

$$x^* = \frac{x}{2D}, \quad y^* = \frac{y}{2D}, \quad u^* = \frac{u}{U}, \quad v^* = \frac{v}{U}, \quad p^* = \frac{p}{\rho U^2}, \quad T^* = \frac{T - T_{in}}{\Delta T}$$

where $2D$ is the characteristic length, U is the characteristic velocity, and $\Delta T = T_{BC} - T_{in}$ is the characteristic temperature difference.

The dimensionless Navier-Stokes equations[11] are then written as:

Dimensionless Continuity Equation:

$$\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} = 0 \tag{8}$$

Dimensionless Momentum Equations:

$$u^* \frac{\partial u^*}{\partial x^*} + v^* \frac{\partial u^*}{\partial y^*} = -\frac{\partial p^*}{\partial x^*} + \frac{1}{Re}\left(\frac{\partial^2 u^*}{\partial x^{*2}} + \frac{\partial^2 u^*}{\partial y^{*2}}\right) \tag{9}$$

$$u^* \frac{\partial v^*}{\partial x^*} + v^* \frac{\partial v^*}{\partial y^*} = -\frac{\partial p^*}{\partial y^*} + \frac{1}{Re}\left(\frac{\partial^2 v^*}{\partial x^{*2}} + \frac{\partial^2 v^*}{\partial y^{*2}}\right) \tag{10}$$

Dimensionless Energy Equation:

$$u^* \frac{\partial T^*}{\partial x^*} + v^* \frac{\partial T^*}{\partial y^*} = \frac{1}{Re \cdot Pr}\left(\frac{\partial^2 T^*}{\partial x^{*2}} + \frac{\partial^2 T^*}{\partial y^{*2}}\right) \tag{11}$$

Here, Re is the Reynolds number, defined as $Re = \frac{\rho U * 2D}{\mu}$, and Pr is the Prandtl number, defined as $Pr = \frac{\mu c_p}{k}$.

3 Physics-Informed Neural Networks

3.1 Model PINNs

Physics-Informed Neural Networks (PINNs) are a novel approach designed to solve problems involving partial differential equations (PDEs). Unlike traditional numerical methods, PINNs use neural networks to approximate the solutions of PDEs by embedding physical knowledge into the training process. Here we model the dimensionless problem, x^* and y^* are dimensionless variables that we use in this 2D model of PINNs (Figure 1) to solve the problem.

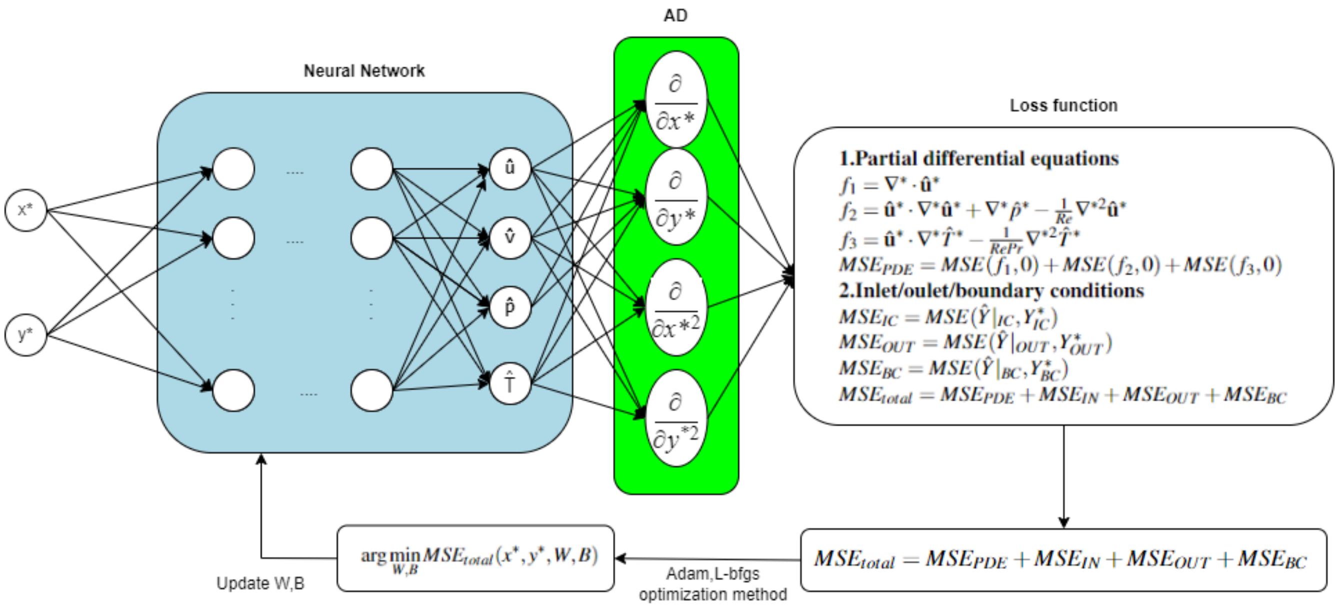

AD
Neural Network
Loss function
x^*
y^*
$\hat{u}$
$\hat{v}$
$\hat{p}$
$\hat{T}$
$\frac{\partial}{\partial x^*}$
$\frac{\partial}{\partial y^*}$
$\frac{\partial}{\partial x^{*2}}$
$\frac{\partial}{\partial y^{*2}}$
1.Partial differential equations
$f_1 = \nabla^* \cdot \hat{\mathbf{u}}^*$
$f_2 = \hat{\mathbf{u}}^* \cdot \nabla^* \hat{\mathbf{u}}^* + \nabla^* \hat{p}^* - \frac{1}{Re} \nabla^{*2} \hat{\mathbf{u}}^*$
$f_3 = \hat{\mathbf{u}}^* \cdot \nabla^* \hat{T}^* - \frac{1}{RePr} \nabla^{*2} \hat{T}^*$
$MSE_{PDE} = MSE(f_1,0) + MSE(f_2,0) + MSE(f_3,0)$
2.Inlet/oulet/boundary conditions
$MSE_{IC} = MSE(\hat{Y}|_{IC}, Y^*_{IC})$
$MSE_{OUT} = MSE(\hat{Y}|_{OUT}, Y^*_{OUT})$
$MSE_{BC} = MSE(\hat{Y}|_{BC}, Y^*_{BC})$
$MSE_{total} = MSE_{PDE} + MSE_{IN} + MSE_{OUT} + MSE_{BC}$
$MSE_{total} = MSE_{PDE} + MSE_{IN} + MSE_{OUT} + MSE_{BC}$
Adam,L-bfgs optimization method
$\arg\min_{W,B} MSE_{total}(x^*, y^*, W, B)$
Update W,B

Figure 1. PINNs model for incompressible fluid flow in two dimensions

Forward Propagation: We can calculate the output values $\hat{u}, \hat{v}, \hat{p}, \hat{T}$ based on the input values x^*, y^* and $\theta = (W, B)$ (Figure 1) of the neural network. Each arrow connecting a neuron in the previous layer to a neuron in the following layer carries a weight value w. Each neuron contains a bias value b. The set of values of w and b forms the matrices W and B. The calculation is performed sequentially from the previous layer to the next layer. Indeed, we want to calculate the value of neurons in the hidden layer h, we call:

- A_{h-1} is the vector representing the values of neurons in the previous layer ($h-1$-th layer).
- $W_{h,i}$ is the weight vector connecting the neuron in the $h-1$ layer to the $Y_{h,i}$ neurons in the h layer.
- $b_{h,i}$ is the bias value of neuron $Y_{h,i}$.
- AF is the activation function of hidden layer.

We can calculate the value of the i-th neuron of layer h by:

$$A_{h,i} = AF(W_{h,i}^T \cdot A_{h-1} + b_{h,i})$$

With the output layer, we do not use the activation function:

$$A_{o,i} = W_{o,i}^T \cdot A_{o-1} + b_{o,i}$$

Then using **automatic differentiation** we can calculate the 1st and 2nd derivatives of $\hat{u}, \hat{v}, \hat{p}, \hat{T}$ (Figure 1). This contributes to calculating the loss function of this neural network. From the PDEs equation and the inlet, outlet and boundary conditions. We move all terms to one side of the equations to set them equal to zero:

$$f_1 := \hat{u}\frac{\partial \hat{u}}{\partial x^*} + \hat{v}\frac{\partial \hat{u}}{\partial y^*} + \frac{\partial \hat{p}}{\partial x^*} - \frac{1}{Re}\left(\frac{\partial^2 \hat{u}}{\partial x^{*2}} + \frac{\partial^2 \hat{u}}{\partial y^{*2}}\right) \qquad f_7 := \hat{T}(-1.25, y^*)$$

$$f_2 := \hat{u}\frac{\partial \hat{v}}{\partial x^*} + \hat{v}\frac{\partial \hat{v}}{\partial y^*} + \frac{\partial \hat{p}}{\partial y^{*2}} - \frac{1}{Re}\left(\frac{\partial^2 \hat{v}}{\partial x^{*2}} + \frac{\partial^2 \hat{v}}{\partial y^{*2}}\right) \qquad f_8 := \hat{p}(x^*, 1.25)$$

$$f_3 := \frac{\partial \hat{u}}{\partial x^*} + \frac{\partial \hat{v}}{\partial y^*} \qquad f_9 := \hat{u}(-0.25, 0.25)$$

$$f_4 := \hat{u}\frac{\partial \hat{T}}{\partial x^*} + \hat{v}\frac{\partial \hat{T}}{\partial y^*} - \frac{1}{Re \cdot Pr}\left(\frac{\partial^2 \hat{T}}{\partial x^{*2}} + \frac{\partial^2 \hat{T}}{\partial y^{*2}}\right) \qquad f_{10} := \hat{v}(-0.25, 0.25)$$

$$f_5 := \hat{u}(-1.25, y^*) - 1 \qquad f_{11} := \hat{T}(-0.25, 0.25) - 1$$

$$f_6 := \hat{v}(-1.25, y^*)$$

The goal is the result of the prediction model that satisfies the PDEs equations and limites conditions. That means the functions f_i get as close to 0 as possible. That's why we define loss functions as:

$$MSE_i = \frac{1}{N_i} \sum_{j=1..N_i} |f_i(x_j^*, y_j^*) - 0|^2$$

Where N_i is the number of points used to train MSE_i. For example, with $i = 5$, f_i will correspond to the inlet condition equation so N_i will be taken with points located at $x = -1$.

Total loss (Figure 1) is calculated as the sum of the loss functions

$$MSE_{total} = \sum_{i=1..11} MSE_i$$

Backward propagation: now we will find W^*, B^* such that they are solutions of the following optimization problem:

$$W^*, B^* = \arg\min_{W,B} MSE_{total}(x^*, y^*, W, B)$$

3.2 Parameter optimization method

The W and B parameters will be continuously updated into the model during the training process. Here to solve this optimal problem of updating weight and bias for the neural network. According to research [12] about optimization methods in PINNs, by combining **Adam** as an initial optimization algorithm, then continuing with **L-BFGS** using the model weights delivered by Adam, the risk of falling into local minima can be reduced. So we will apply **L-BFGS** after the total loss value using the **Adam** optimization method does not decrease anymore. Both optimization methods are readily available in the deepXDE library[9]; however, we will briefly describe these two methods below.

3.2.1 Adam algorithm

The **Adam** algorithm is a first-order gradient-based optimization of stochastic loss functions and has enjoyed widespread application to fitting large and complex deep neural networks [12]. It combines the advantages of **Momentum** and **RMSProp** to achieve high and stable convergence efficiency:

- **RMSprop** calculates the mean square of the gradient in the previous steps, then divides the current gradient by the square root of this value. This reduces the size of the gradient when it is large and increases the size when it is small, leading to more stable convergence:

$$S_{dW} = \beta_2 \cdot S_{dW-prev} + (1-\beta_2) \cdot (dW)^2$$

$$S_{dB} = \beta_2 \cdot S_{dB-prev} + (1-\beta_2) \cdot (dB)^2$$

$$W = W - \alpha \cdot \frac{dW}{\sqrt{S_{dW}} + \varepsilon}$$

$$B = B - \alpha \cdot \frac{dB}{\sqrt{S_{dB}} + \varepsilon}$$

- The **Momentum** Optimizer accumulates gradients at each step by adding a portion of the previous gradient to the current gradient. This creates momentum for gradient descent, helping it move faster in the direction of the steepest descent

$$V_{dW} = \beta_1 \cdot V_{dW-prev} + (1-\beta_1) \cdot dW$$

$$V_{dB} = \beta_1 \cdot V_{dB-prev} + (1-\beta_1) \cdot dB$$

$$W = W - \alpha \cdot V_{dW}$$

$$B = B - \alpha \cdot V_{dB}$$

- **Adam** combines the advantages of both the fast convergence speed of **Momentum** and the stability of **RMSprop**:

$$W = W - \alpha \cdot \left(\frac{V_{dW}}{\sqrt{S_{dW}} + \varepsilon} \right)$$

$$B = B - \alpha \cdot \left(\frac{V_{dB}}{\sqrt{S_{dB}} + \varepsilon} \right)$$

Here, we use the optimization function available in Deepxde [9] with default parametre: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e-7$.

3.2.2 Limited-memory BFGS algorithm

Limited-memory BFGS (L-BFGS or LM-BFGS) is an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) using a limited amount of computer memory. This algorithm is particularly effective for large-scale optimization problems [13].

The most popular Quasi-Newton algorithm is the BFGS method, named for its discoverers, Broyden, Fletcher, Goldfarb, and Shanno [14]. In this method, α_k is chosen to satisfy the Wolfe Condition [15] so there is no need to manually select a constant value for α_k. The step direction p_k is calculated based on an approximation of the inverse of the Hessian matrix. In each iteration, the approximation of the inverse of the Hessian matrix is updated based on the current s_k and y_k values, where s_k presents the position difference and y_k represents the gradient difference in the iteration. These vectors are the same length as vector x.

BFGS needs to keep an approximation of the inverse of the Hessian matrix in each iteration (an $n \times n$ matrix), where n is the length of the parameter vector x. It becomes infeasible to store this matrix in memory for large values of n[16].

The L-BFGS (Limited-memory BFGS) algorithm modifies BFGS to obtain Hessian approximations that can be stored in just a few vectors of length n. Instead of storing a fully dense $n \times n$ approximation, L-BFGS stores just m vectors ($m \ll n$) of length n that implicitly represent the approximation. The main idea is that it uses curvature information from the most recent iterations. The curvature information from earlier iterations is considered to be less likely to be relevant to the Hessian behavior at the current iteration and is discarded in favor of the memory[16].

In L-BFGS, the $\{s_k, y_k\}$ pairs are stored from the last m iterations, which causes the algorithm to need $2 \times m \times n$ storage compared to $n \times n$ storage in the BFGS algorithm. The $2 \times m$ memory vectors, along with the gradient at the current point, are used in the L-BFGS two-loop recursion algorithm to calculate the step direction. The L-BFGS algorithm and its two-loop recursion are shown in Algorithms 1 and 2, respectively. The Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm is designed to solve large-scale optimization problems by approximating the BFGS algorithm using a limited amount of computer memory. The algorithm can be described as follows[16]:

Algorithm 1 L-BFGS

```
 1: procedure L-BFGS
 2:     Choose starting point $x_0$, and integer $m > 0$
 3:     $k \leftarrow 0$
 4:     while true do
 5:         Calculate $\Delta f(x_k)$ at the current point $x_k$
 6:         Calculate $p_k$ using Algorithm 2
 7:         Calculate $\alpha_k$ where it satisfies Wolfe conditions
 8:         $x_{k+1} \leftarrow x_k + \alpha_k p_k$
 9:         if $k > m$ then
10:             Discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from storage
11:         end if
12:         Compute and Save $s_k = x_{k+1} - x_k$ and $y_k = \Delta f_{k+1} - \Delta f_k$
13:         $k \leftarrow k + 1$
14:     end while
15: end procedure
```

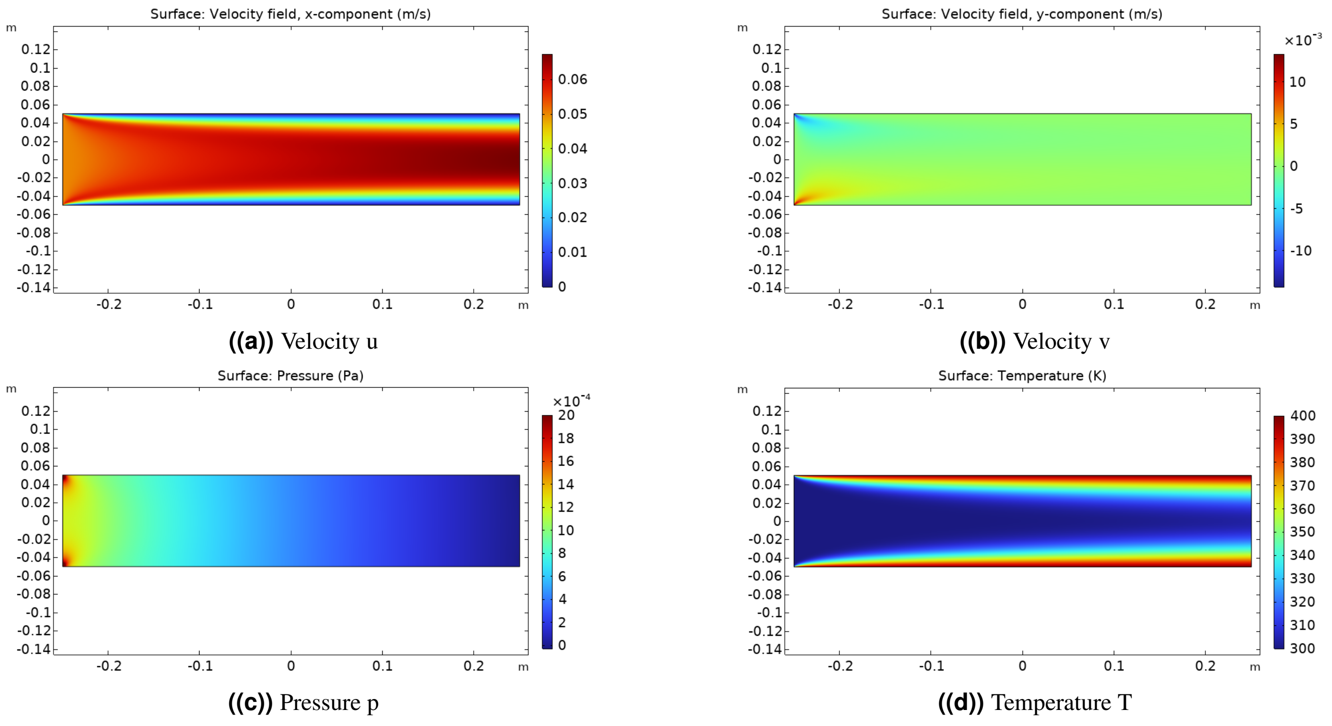
Algorithm 2 L-BFGS Two-loop Recursion

```
1:  procedure TWO-LOOP RECURSION
2:      $q \leftarrow -\Delta f(x_k)$
3:      for $i = k-1, k-2, \dots, k-m$ do
4:          $\alpha_i \leftarrow \frac{s_i \cdot q}{y_i \cdot s_i}$
5:          $q \leftarrow q - \alpha_i y_i$
6:      end for
7:      $r \leftarrow \frac{s_{k-1} \cdot y_{k-1}}{y_{k-1} \cdot y_{k-1}} q$
8:      for $i = k-m, k-m+1, \dots, k-1$ do
9:          $\beta \leftarrow \frac{y_i \cdot r}{y_i \cdot s_i}$
10:         $r \leftarrow r + s_i(\alpha_i - \beta)$
11:     end for
12:     return $r$
13: end procedure
```

4 Experiment Setup

For the implementation of the experiment, the code is available on GitHub (Appendix A). In this study, we focus on the airflow in a 2D rectangular model between two plates with a length of 0.5m and a gap of 0.1m. Below is an image depicting the steady airflow of air obtained from the CFD solver (Appendix A). With air parameters:

- $rho = 1.2\ kg/m^3$
- $mu = 1.8 * 10^5\ Pa.s$
- $c_p = 1005\ J/(kg.K)$
- $l = 0.0257\ W/(m.K)$



((a)) Velocity u **((b))** Velocity v **((c))** Pressure p **((d))** Temperature T

Figure 2. Results according to CFD solver.

And the limit conditions:

- **Inlet conditions:** $u_{in} = 0.05m/s, v_{in} = 0m/s, T_{in} = 300K$
- **Outlet conditions:** $p_{out} = 0$ Pa
- **Boundary conditions:** $u_{bc} = 0m/s, v_{bc} = 0m/s, T_{bc} = 400K$

Observations:

- **Velocity** u: The first image shows the horizontal velocity component (u). The highest velocity (red) is concentrated in the center, gradually decreasing towards the plates (blue) and reaching zero at the two plates
- **Velocity** v: The second image shows the vertical velocity component (v). The velocity in this direction is much smaller than in the horizontal direction and is almost zero throughout the space between the plates.
- **Pressure** p: The third image shows the pressure distribution in the space between the two flat plates. The pressure decreases gradually from the inlet (red) to the outlet (blue) of this space.
- **Temperature** T: The final image shows the temperature distribution in the space between the two flat plates. The highest temperature (red) is concentrated near the surfaces of the two plates, gradually decreasing towards the center (blue).

4.1 Build and train the PINNs model

First, we randomly select train points between 2 plates and at condition limites. With the number of training points being $N_{PDE} = 1000$-points located between 2 plates to calculate loss function of PDEs, $N_{BC} = 500$-points located on 2 plates to calculate loss function of boundary conditions, $N_{IN} = 200$ and $N_{OUT} = 200$-points located on inlet/outlet to calculate loss function inlet/oulet conditions.

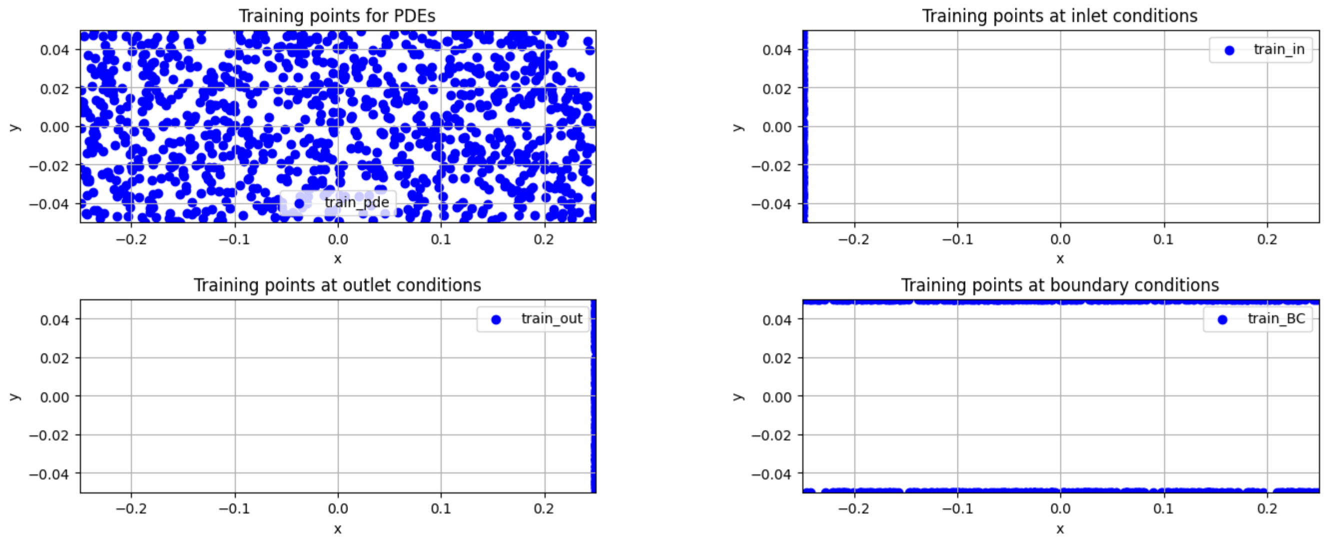
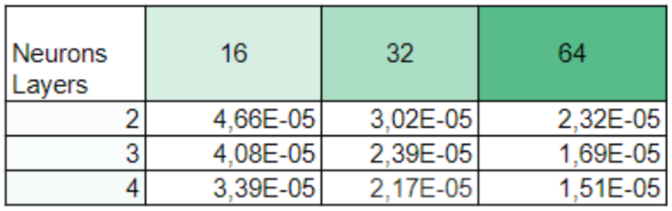


Figure 3. Training points

Then we perform training with the number of hidden layers from 2 to 4 and the number of neurons in each hidden layer from 16,32 and 64. We have the following comparison table:



Neurons Layers	16	32	64
2	4,66E-05	3,02E-05	2,32E-05
3	4,08E-05	2,39E-05	1,69E-05
4	3,39E-05	2,17E-05	1,51E-05

Figure 4. Comparison of Total Loss for Models with Varying Hidden Layers and Neurons.

Increasing the number of neurons in each layer decreases the total loss. This is because a larger network has more capacity to learn complex patterns. Therefore, we choose 4 hidden layers with 64 neurons per hidden layer to give us the best results.

Additionally, we chose the **tanh activation function** because it is smooth and differentiable, essential for accurately approximating solutions of differential equations. The tanh function maps input values to a range between -1 and 1, helping normalize data and facilitating effective training by avoiding issues with unbalanced gradients. Furthermore, it provides stronger gradients for inputs near zero, unlike the sigmoid function, aiding efficient training. Its zero-centered nature also helps mitigate internal covariate shift, leading to faster convergence during training [17].

Optimization methods:

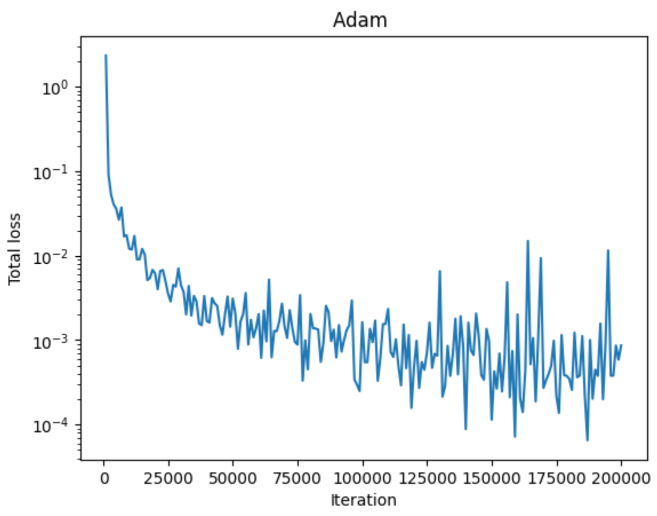


Figure 5. Training by Adam (200000 steps)

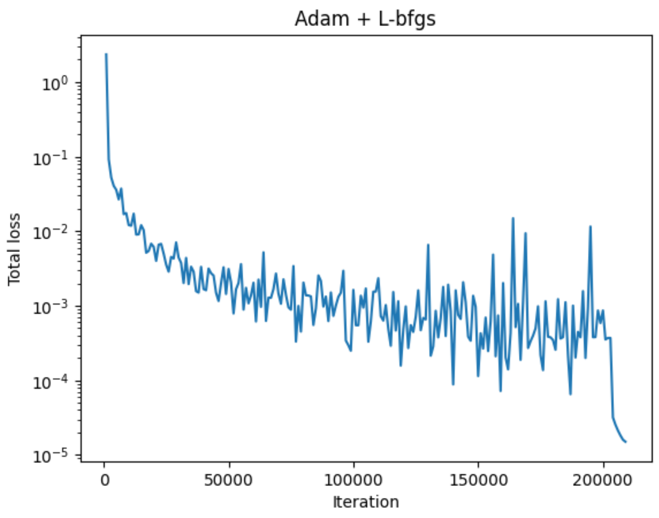


Figure 6. Training by Adam and L-BFGS (205542 steps)

First, we trained this model using the Adam optimization method (Figure 5). As we can see, Adam helps reduce the total loss rapidly and approach close to the global minimum. However, from 150k iterations onward, the total loss fluctuates within a certain range and does not continue to decrease. Next, we continued training by L-BFGS for 5542 iterations (Figure 6). It can be observed that the total loss can continue to decrease significantly and reaches a minimum at the 205542nd step. Thus, the combination of these two optimization methods works effectively with this model.

5 Result and discussion

The PINNs model is trained to predict the dimensionless model. After calculating the results from the dimensionless model, apply the above formulas to convert each variable to the corresponding dimensional value:

$$x = x^*.2D, \quad y = y^*.2D, \quad u = u^*.U, \quad v = v^*.U, p = p^*.\rho U^2, \quad T = T^*.\Delta T + T_{in}$$

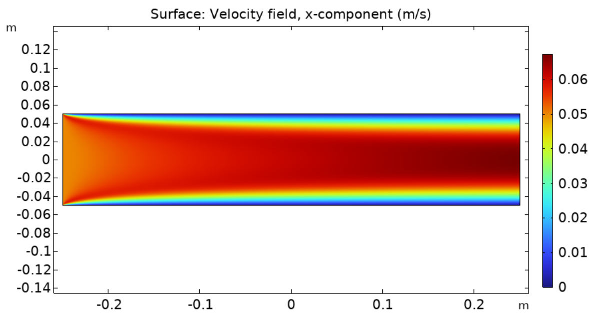


Figure 7. Velocity u (CFD)

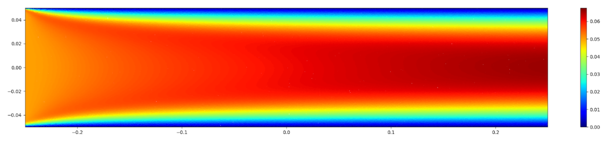


Figure 8. Velocity u (PINNs)

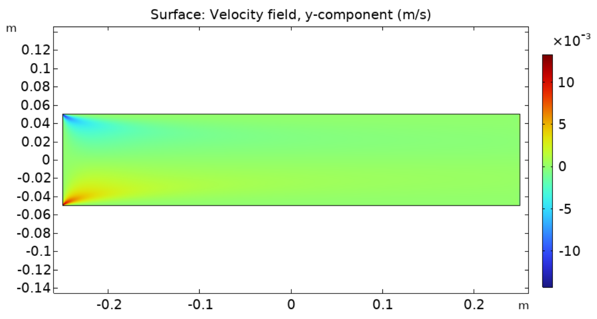


Figure 9. Velocity v (CFD)

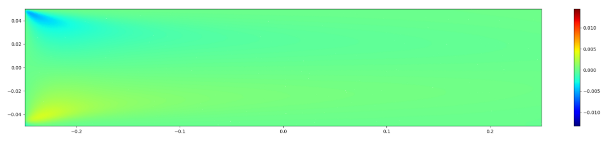


Figure 10. Velocity v (PINNs)

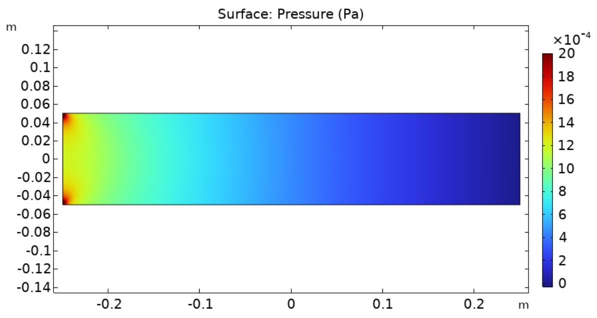


Figure 11. Pressure p (CFD)

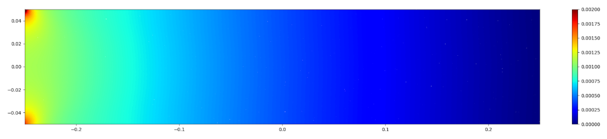


Figure 12. Pressure p (PINNs)

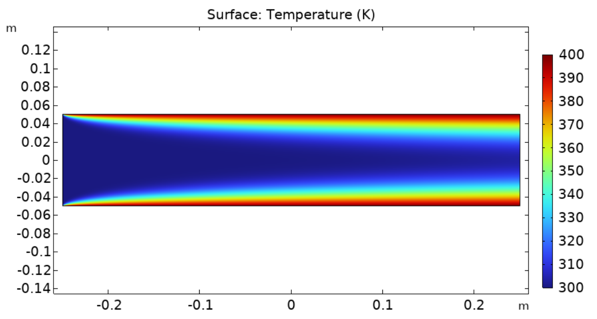


Figure 13. Temperature T (CFD)

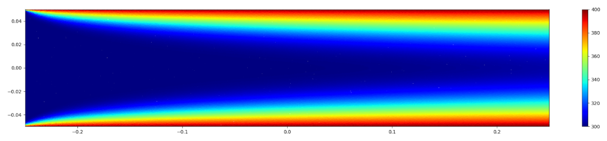


Figure 14. Temperature T (PINNs)

Figure 15. Comparison of results by CFD and PINNs

From the results obtained by CFD and PINNs (Figure 15) we have the following observations:

Similarities:

- **Overall Shape:** Both results exhibit a similar overall shape, especially in the stable flow region in the middle of the two plates (Figure 15).
- **Trends:** Both models demonstrate:
  - The velocity u is highest in the center and gradually decreases to 0 on both sides of the plates (Figure 7 and Figure 8).
  - The velocity v is zero at almost every point except the two corners at the inlet (Figure 9 and Figure 10).
  - A pressure decrease along the length of the channel (Figure 11 and Figure 12).
  - A temperature distribution corresponding to the flow pattern (Figure 13 and Figure 14).

Differences:

- **Velocity u:** The velocity u between the plates in the PINNs model is slightly lower (indicated by a lighter red) compared to the CFD (Figure 7 and Figure 8).
- **Velocity v:** In the bottom left corner, corresponding to the corner of the lower plate and the inlet, the velocity v in the PINNs model is lower (lighter yellow) compared to the CFD (Figure 9 and Figure 10).
- **Pressure p:** Similarly, in the two corners at the inlet, the pressure p in the PINNs model is lower compared to the CFD (Figure 11 and Figure 12) .

Interpretation:
While both PINNs and CFD (Appendix A) provide similar overall flow patterns and trends, there are some differences in the detailed velocity and pressure fields (Figure 15). Specifically, the PINNs model shows slightly lower velocity and pressure values in certain regions compared to the CFD (Appendix A). These differences may be because the model is not fully optimized (total loss is not small enough). However, the error is small and completely acceptable so PINNs work well to solve the Navier-Stokes and heat balance equations in this case.

6 Conclusion

In this study, we successfully demonstrated the application of Physics-Informed Neural Networks (PINNs) to simulate the behavior of a fixed-bed catalytic reactor. By integrating the Navier-Stokes equations and the heat balance equation into the neural network's loss function, we effectively guided the model to learn from the governing physics of the problem. This approach allowed us to accurately predict temperature, pressure, and velocity distributions within the air flow model, showcasing the potential of PINNs in solving complex PDEs with high precision.

Our results show that PINNs are highly accurate in solving this fluid dynamics and heat transfer simulations problem. The dimensionless form of the Navier-Stokes equations facilitated a more efficient optimization process, leading to more reliable solutions. Furthermore, using dimensionless model allows us to easily substitute different parameters corresponding to various fluids, gases, lengths of the fixed bed catalytic reactor and boundary conditions without the need to rebuild the PINNs model. Additionally, the combination of the Adam optimizer with the L-BFGS method proved effective in reducing the risk of falling into local minima and achieving better convergence.

The validation of our PINNs model against numerical simulations confirmed the possibility of satisfaction with its basic scientific laws.. The improved predictive accuracy and computational efficiency demonstrated by this approach suggest that PINNs hold great promise for broader applications in computational physics and engineering, particularly in systems characterized by complex interactions of physical and chemical processes and systems where data collection takes a lot of time.

Future work may focus on extending this methodology to three-dimensional models and exploring its applicability to other types of reactors and physical systems. Additionally, we can develop a more effective optimization method to minimize the total loss of the model, thereby providing more accurate results. Furthermore, integrating real experimental data could further validate and enhance the robustness of the PINNs approach. Overall, this research highlights the potential of leveraging machine learning and neural networks to tackle intricate problems in sience and beyond.

References

[1] 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. *J. Comput. Phys.* **378**, 686–707, DOI: https://doi.org/10.1016/j.jcp.2018.10.045 (2019).

[2] Chen, Z., Liu, Y. & Sun, H. Physics-informed learning of governing equations from scarce data. *Nat. Commun.* **12**, 6136, DOI: 10.1038/s41467-021-26434-1 (2021).

[3] Kharazmi, E., Zhang, Z. & Karniadakis, G. E. Variational physics-informed neural networks for solving partial differential equations. *CoRR* **abs/1912.00873** (2019). 1912.00873.

[4] Yan, J., Chen, X., Wang, Z., Zhoui, E. & Liu, J. St-pinn: A self-training physics-informed neural network for partial differential equations. In *2023 International Joint Conference on Neural Networks (IJCNN)*, 1–8, DOI: 10.1109/IJCNN54540.2023.10191472 (2023).

[5] Cammi, A., Marcello, V. D., Luzzi, L., Memoli, V. & Ricotti, M. A multi-physics modelling approach to the dynamics of molten salt reactors. *Annals Nucl. Energy* **38**, 1356–1372, DOI: 10.1016/J.ANUCENE.2011.01.037 (2011).

[6] Logtenberg, S., Nijemeisland, M. & Dixon, A. Computational fluid dynamics simulations of fluid flow and heat transfer at the wall–particle contact points in a fixed-bed reactor. *Chem. Eng. Sci.* **54**, 2433–2439, DOI: https://doi.org/10.1016/S0009-2509(98)00445-X (1999).

[7] Shahrokhi, M. & Baghmisheh, G. Modeling, simulation and control of a methanol synthesis fixed-bed reactor. *Chem. Eng. Sci.* **60**, 4275–4286, DOI: 10.1016/J.CES.2004.12.051 (2005).

[8] Markidis, S. The old and the new: Can physics-informed deep-learning replace traditional linear solvers? *Front. Big Data* **4**, DOI: 10.3389/fdata.2021.669097 (2021).

[9] Lu, L., Meng, X., Mao, Z. & Karniadakis, G. E. Deepxde: A deep learning library for solving differential equations. *SIAM Rev.* **63**, 208–228, DOI: 10.1137/19M1274067 (2021). https://doi.org/10.1137/19M1274067.

[10] Quarteroni, A. & Valli, A. The steady navier-stokes problem. In *Numerical Approximation of Partial Differential Equations*, 339–362 (Springer, 1994).

[11] Doering, C. R. & Gibbon, J. D. *Applied analysis of the Navier-Stokes equations*. 12 (Cambridge University Press, 1995).

[12] Taylor, J., Wang, W., Bala, B. & Bednarz, T. Optimizing the optimizer for data driven deep neural networks and physics informed neural networks. (2022).

[13] Liu, D. C. & Nocedal, J. On the limited memory bfgs method for large scale optimization. *Math. Program.* **45**, 503–528 (1989).

[14] Nocedal, J. & Wright, S. J. *Numerical Optimization* (Springer, New York, 2006), 2nd edn.

[15] Wolfe, P. Convergence conditions for ascent methods. *SIAM Rev.* **11**, 226–235 (1969).

[16] Najafabadi, M. M., Khoshgoftaar, T. M., Villanustre, F. & Holt, J. Large-scale distributed l-bfgs. *J. Big Data* **4**, 22, DOI: 10.1186/s40537-017-0084-5 (2017).

[17] Dung, D. V., Song, N. D., Palar, P. S. & Zuhal, L. R. On the choice of activation functions in physics-informed neural network for solving incompressible fluid flows. In *AIAA SCITECH 2023 Forum*, 1803, DOI: 10.2514/6.2023-1803 (2023).

A Appendix

In this study, we used Deepxde [9] as the framework to build PINNs model. Here, Deepxde uses TensorFlow as the default backend.

To evaluate the results obtained from PINNs, we use CFD simulations for a 2D geometry with software Comsol Multiphysics. The numerical method used by the software is the finite element method.

Details on how to build PINNS model can be found in the source code of this study at: `https://github.com/maidinhnam/PINNs`