Apply Physics Informed Neural Network to Solve Navier–Stokes equations

Ning Deng
Weiyang College
dengn22@mails.tsinghua.edu.cn

Gongshu Liu
Department of Electronic Engineering
liugs22@mails.tsinghua.edu.cn

Bowen Ren
Zhili College
rbw21@mails.tsinghua.edu.cn

Abstract

This paper discusses the application of Physics Informed Neural Networks (PINNs) in solving two-dimensional steady-state Navier-Stokes equations, with a focus on three classic fluid flow problems: Poiseuille flow, Kovasznay flow, and cavity flow. To address issues such as slow convergence, low precision, and gradient vanishing or exploding encountered in solving the cavity flow problem, a series of improvements were made, such as introducing the stream function term and adaptive weights into the loss function. These measures significantly enhanced the model's stability and solution accuracy. By incorporating the stream function term, the model better satisfies boundary conditions and fluid continuity requirements, resolving issues related to gradient explosion and disappearance. Meanwhile, the use of adaptive weights effectively balanced the importance of different parts in the loss function, allowing the network to focus more on those parts that converge slowly or are difficult to optimize, thus speeding up overall convergence and improving model precision.

1. Background

The Navier–Stokes equations(NSE)are partial differential equations which describe the motion of viscous fluid substances. Classical numerical methods, such as the finite difference method and finite element method(FEM), can produce an approximate solution for the NSE. However, deep learning techniques have recently become popular for solving PDEs. One such method is the physics informed neural network (PINN).

1.1. Two-dimensional Steady Continuous Navier-Stokes Equations

The Navier-Stokes (NS) equations solved in this text are all two-dimensional steady-state flow fields of NS equations, which take the following form:

$$\begin{cases} \dfrac{\partial u}{\partial x}+\dfrac{\partial v}{\partial y}=0 \\ u\dfrac{\partial u}{\partial x}+v\dfrac{\partial u}{\partial y}+\rho\dfrac{\partial p}{\partial x}-\nu(\dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2})=0 \\ u\dfrac{\partial v}{\partial x}+v\dfrac{\partial v}{\partial y}+\rho\dfrac{\partial p}{\partial y}-\nu(\dfrac{\partial^2 v}{\partial x^2}+\dfrac{\partial^2 v}{\partial y^2})=0 \end{cases}$$

In the context of the Navier-Stokes equations, where u represents the velocity component of the flow field in the x-direction, v is the velocity component in the y-direction, p denotes the pressure of the flow field, $\rho$is the density of the fluid, and ν is the kinematic viscosity. Throughout our solution, we arbitrarily choose $\rho = 1$ and $\nu = 0.01$, resulting in a Reynolds number (Re) of 100. Consequently, we adopt the following form of the equations for subsequent discussion and coding:

$$\begin{cases} \dfrac{\partial u}{\partial x}+\dfrac{\partial v}{\partial y}=0 \\ u\dfrac{\partial u}{\partial x}+v\dfrac{\partial u}{\partial y}+\dfrac{\partial p}{\partial x}-\dfrac{1}{Re}(\dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2})=0 \\ u\dfrac{\partial v}{\partial x}+v\dfrac{\partial v}{\partial y}+\dfrac{\partial p}{\partial y}-\dfrac{1}{Re}(\dfrac{\partial^2 v}{\partial x^2}+\dfrac{\partial^2 v}{\partial y^2})=0 \\ where Re=100 \end{cases}$$

1.2. Boundary Conditions

This paper selects three classical flows as computational examples.The solution for all examples will be limited to $[0,1]\times[0,1]$.
The first example is the simplest Poiseuille flow. We consider laminar flow in a two-dimensional pipe:

$$u=\begin{cases} y(1-y) & \text{at left and right boundaries} \\ 0 & \text{at other boundaries} \end{cases}$$

$$v = 0 \text{at all the boundaries}$$

The second example is the Kovaszney flow, which is an analytical solution to the Navier-Stokes equations, frequently used to test the performance of Navier-Stokes equation solvers, and takes the following form:

$$\begin{aligned} u &= 1 - e^{\lambda x}\cos 2\pi y \\ v &= \frac{\lambda}{2\pi}e^{\lambda x}\sin 2\pi y \\ p &= \frac{1}{2}(1 - 2e^{2\lambda x}) \\ where \lambda &= \frac{Re}{2} - \sqrt{\frac{Re^2}{2} + 4\pi^2} \end{aligned}$$

The third example is the cavity flow, a highly nonlinear flow that can challenge the stability of Navier-Stokes equation solvers. Its boundary conditions are as follows:

$$u = \begin{cases} 1 & \text{at top boundary} \\ 0 & \text{at other boundaries} \end{cases}$$

$$v = 0 \text{at all the boundaries}$$

1.3. Traditional model

In the solution of Examples 1 and 2, this paper employs the same neural network as used in Rassi's original paper. The input of the neural network is set to coordinates (x, y), and the output is set to stream function and pressure (ϕ, p), where $\frac{\partial \phi}{\partial y} = u, -\frac{\partial \phi}{\partial x} = v$. The neural network consists of four hidden linear layers, each with a size of 32x32, and employs the SiLU activation function.

In Example 1, 3200 points are randomly sampled along each boundary and 12800 points are randomly sampled within the region at each epoch. The neural network generates the stream function ϕ and pressure p. The average loss is computed on the boundaries (comparing to the boundary conditions) and within the region (comparing to the equation) using the logcosh loss function. Subsequently, optimization is performed using the Adam optimizer. The overall loss function takes the following form:

$$\begin{aligned} loss_{boundary} &= \frac{1}{N}\sum_{1}^{N}\log\cosh(u - utarget) \\ &+ \frac{1}{N}\sum_{1}^{N}\log\cosh(v - v_{target}) \\ loss_{central} &= \frac{1}{N}\sum_{1}^{N}\log\cosh(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} \\ &- \frac{1}{Re}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})) \\ &+ \frac{1}{N}\sum_{1}^{N}\log\cosh(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} \\ &- \frac{1}{Re}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2})) \\ loss_{total} &= loss_{boundary} + loss_{central} \end{aligned}$$

In Example 2, initially, 51,200 points are sampled within the region. During each epoch, an additional 12,800 points are randomly selected within the region. These points are processed through a neural network to obtain values for ϕ and p. The average loss is calculated for both the predetermined points and the randomly sampled points by comparing them to the analytical solution and the formula, respectively. This is done using the cosh loss function. Subsequently, optimization is performed using the Adam optimizer. The overall loss function is similar to that in Example 1.

1.4. Result and Reflection

The fundamental PINN model is capable of resolving these two examples, as can be seen. However, in Example 1, the value of v is slightly overestimated, and in Example 2, the value of u deviates from the analytical solution near the upper and lower boundaries. This is primarily due to the fact that in these regions, the value of $loss_{central}$ is significantly smaller than $loss_{boundary}$, leading to the averaging process overwhelming $loss_{boundary}$, which in turn results in incomplete optimization of $loss_{boundary}$ and a less accurate alignment with the boundary conditions. This issue becomes more severe in the resolution of flow in a square cavity, and a potential solution to this problem will be discussed later in this text.

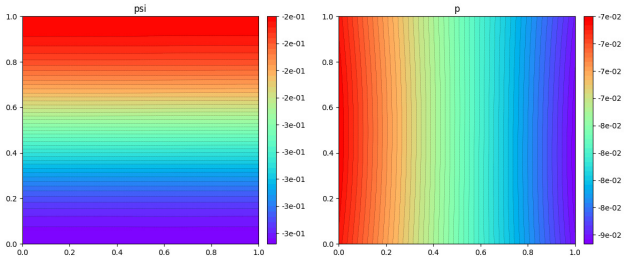

Figure 1. phi and p in example 1

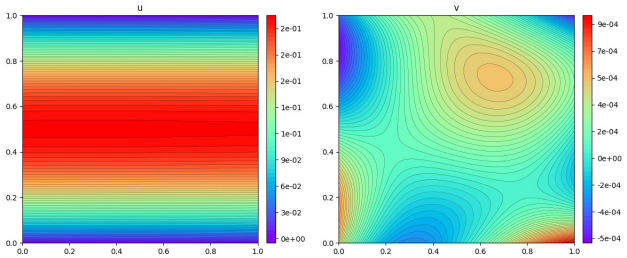

Figure 2. u and v in example 1

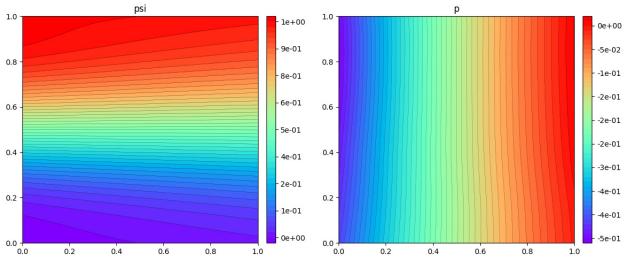

Figure 3. phi and p in example 2

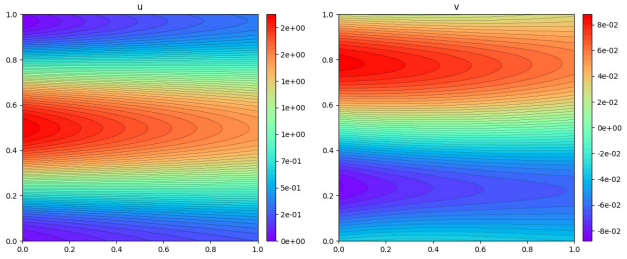

Figure 4. u and v in example 2

2. Improvement of the loss function

In PINN, the loss function typically comprises four parts: the initial conditions, boundary conditions, equation residuals, and real data points. However, for the cavity flow problem of the NS equation solved in this project, the loss function will only include boundary conditions and equation residuals. This is because the cavity flow is a steady-state problem, and we have not employed a data-driven approach.

$$L_{PDE}(\theta; T_f) = \frac{1}{|T_f|} \sum_{x \in T_f} \left\| f\left(x; \frac{\partial \hat{u}}{\partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_d}, \ldots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}\right) \right\|_2^2$$

$$L_{IC}(\theta; T_i) = \frac{1}{|T_i|} \sum_{x \in T_i} \|\hat{u}(x) - u(x)\|_2^2$$

$$L_{BC}(\theta; T_b) = \frac{1}{|T_b|} \sum_{x \in T_b} \|B(\hat{u}, x)\|_2^2$$

$$L_{Data}(\theta; T_{data}) = \frac{1}{|T_{data}|} \sum_{x \in T_{data}} \|\hat{u}(x) - u(x)\|_2^2$$

In the specific solving process, we let the PINN network produce two outputs, representing the stream function and pressure, respectively. Here we introduce the stream function $\psi(x, y)$. The stream function is a scalar function commonly used to describe two-dimensional, incompressible, and steady fluid flow.

- The x-component of fluid motion = the partial derivative of some stream function with respect to y, i.e., $u = \frac{\partial \psi}{\partial y}$.
- The y-component of fluid motion = the negative partial derivative of some stream function with respect to x, i.e., $v = -\frac{\partial \psi}{\partial x}$.

The advantage of this representation is that it automatically satisfies the requirement of the continuity equation.

Although the overall structure of the loss function has been established, selecting which and how many physical quantities to represent the loss function is a challenging problem. Initially, we only used the velocity components u and v to compute the loss on the boundary conditions. However, after numerous experiments, we found that such a loss function not only converged slowly but also often experienced issues with gradient explosion or vanishing. Initially, we suspected that the activation function was causing numerical instability, which in turn led to gradient vanishing or explosion. Therefore, we chose the smoother activation function SiLU, but the issue still did not improve. Eventually, we even doubted whether the cavity flow problem, with its discontinuous boundary conditions, could be solved using PINNs.

After consulting several pieces of literature, we realized that adding physical quantities to the loss function could effectively address the issues of numerical instability that caused gradient explosion or vanishing. Consequently, we added the stream function term to the loss function for the boundary conditions and successfully resolved the problems of gradient vanishing or explosion. Upon further analysis, we believe that prior to the inclusion of the stream function, the neural network may have been trying to satisfy the velocity conditions on the boundaries by adjusting

the internal flow field, thus ignoring the global consistency of the flow field and leading to numerical instability. We had considered incorporating pressure boundary conditions, but due to difficulties in obtaining accurate pressure boundary conditions, we ultimately only used the stream function ψ.

3. Self-adaptive physics-informed neural networks

For the loss function, we noticed that there are several items. While training, it happens that some items converge to be small, but others are still large. Hence, we used Self-adaptive physics-informed neural networks. By taking this method, we give the items that are large more weights, and the items that are small less, to help the machine focus more on the large items and find the correct descending direction. Besides, in the original loss function, different terms represent the discrepancy between the numerical solutions and the true solutions across various dimensions. Initially, we simply summed these different terms. However, in reality, the same numerical value may reflect varying degrees of discrepancy for these terms. Therefore, at the beginning of training, we assign distinct weight functions to each term.

The proposed self-adaptive PINN utilizes the following loss function

$$L_r(w,\lambda_r)=\sum_{i=1}^{N_i} m(\lambda_r^i)[\mathcal{N}_i(u(x_i,t_i;w))-f(x_i,t_i)]^2 \quad (1)$$

$$L_b(w,\lambda_b)=\sum_{i=1}^{N_b} m(\lambda_b^i)[\mathcal{B}_i(u(x_b,t_i;w))-g(x_b,t_b^i)]^2 \quad (2)$$

$$L_0(w,\lambda_0)=\sum_{i=1}^{N_0} m(\lambda_0^i)[u(x_0;w)-h(x_0)]^2 \quad (3)$$

where $\lambda_r = (\lambda_r^1,\dots,\lambda_r^{N_r})$, $\lambda_b = (\lambda_b^1,\dots,\lambda_b^{N_b})$, and $\lambda_0 = (\lambda_0^1,\dots,\lambda_0^{N_0})$ are trainable, nonnegative self-adaptation weights for the initial, boundary, and residue points, respectively.

The self-adaptation mask function m defined on $(0,\infty)$ is a nonnegative, differentiable on $(0,+\infty)$, strictly increasing function of λ. A key feature of self-adaptive PINNs is that the loss L is minimized with respect to the network weights w, as usual, but is maximized with respect to the self-adaptation weights λ. The corresponding gradient descent/ascent steps are:

$$w^{k+1}=w^k-\eta\nabla_w L(w^k,\lambda^k,\lambda_b^k,\lambda_0^k) \quad (4)$$

$$\lambda^{k+1}=\lambda^k+\rho\nabla_\lambda L(w^k,\lambda^k,\lambda_b^k,\lambda_0^k) \quad (5)$$

$$\lambda_b^{k+1}=\lambda_b^k+\rho\nabla_{\lambda_b} L(w^k,\lambda^k,\lambda_b^k,\lambda_0^k) \quad (6)$$

$$\lambda_0^{k+1}=\lambda_0^k+\rho\nabla_{\lambda_0} L(w^k,\lambda^k,\lambda_b^k,\lambda_0^k) \quad (7)$$

where $\eta>0$ is the learning rate for the neural network weights at step k, and $\rho>0$ is a separate learning rate for the self-adaptation weights.

$$\nabla_{\lambda_r} L(w^k,\lambda_r^k,\lambda_b^k,\lambda_0^k)=\frac{1}{2}\begin{bmatrix} m'(\lambda_r^{k,1})[\mathcal{N}_r(u(x_r,t_r;w^k))-f(x_r^{k,N_r},t_r)]^2 \\ \vdots \\ m'(\lambda_r^{k,N_r})[\mathcal{N}_r(u(x_r,t_r;w^k))-f(x_r^{k,N_r},t_r)]^2 \end{bmatrix} \quad (8)$$

$$\nabla_{\lambda_b} L(w^k,\lambda_r^k,\lambda_b^k,\lambda_0^k)=\frac{1}{2}\begin{bmatrix} m'(\lambda_b^{k,1})[\mathcal{B}_b(u(x_b,t_b;w^k))-g(x_b^{k,1},t_b^k)]^2 \\ \vdots \\ m'(\lambda_b^{k,N_b})[\mathcal{B}_b(u(x_b,t_b;w^k))-g(x_b^{k,N_b},t_b^k)]^2 \end{bmatrix} \quad (9)$$

$$\nabla_{\lambda_0} L(w^k,\lambda_r^k,\lambda_b^k,\lambda_0^k)=\frac{1}{2}\begin{bmatrix} m'(\lambda_0^{k,1})[u(x_0;w^k)-h(x_0^{k,1})]^2 \\ \vdots \\ m'(\lambda_0^{k,N_0})[u(x_0;w^k)-h(x_0^{k,N_0})]^2 \end{bmatrix} \quad (10)$$

Hence, since $m'(\lambda)>0$ (the mask function is strictly increasing, by assumption), then $\nabla_\lambda L$, $\nabla_{\lambda_b} L$, $\nabla_{\lambda_0} L>0$, and any gradient component is zero if and only if the corresponding unmasked loss is zero. For example, we may set the mask function $m(\lambda)=c\lambda^q$, for $c,q>0$. Then, if some items of the loss function are larger, the respective gradient of λ is larger, leading to an increase in the respective λ.

4. Result

We applied the improved PINN model to solve the cavity flow problem, and after 10,000 epochs, we were able to achieve relatively good fitting results.

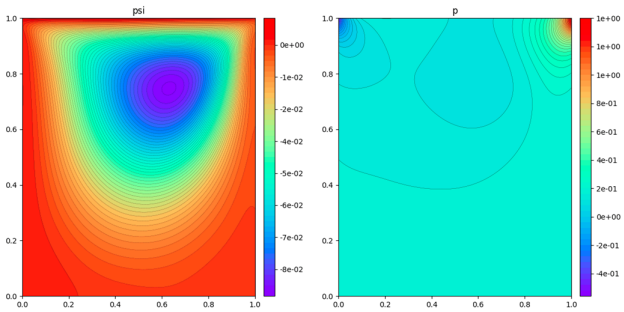

Figure 5. phi and u of Cavity Flow Predicted by PINN

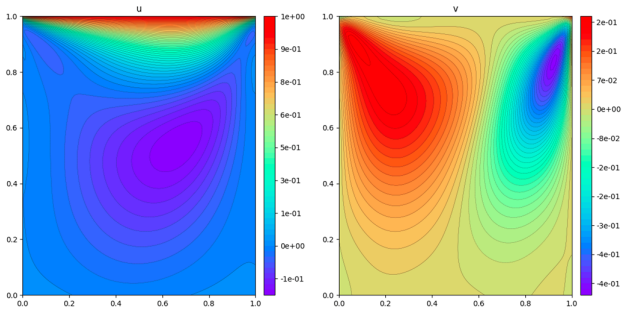

Figure 6. v and p of Cavity Flow Predicted by PINN

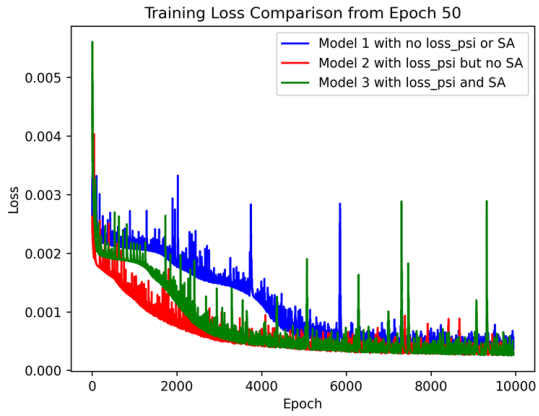

Figure 7. Comparison of Training Loss Across Three Models

Furthermore, we compared three models: the original model, the model with the addition of the stream function loss but without adaptive weights, and the model with both the stream function loss and adaptive weights. We found that both the second and third models had faster convergence rates compared to the first model. Additionally, when the number of epochs was sufficiently large, the accuracy of the third model was relatively higher.

5. Reference

[1]Levi D. McClenny, Ulisses M. Braga-Neto (2023).Self-adaptive physics-informed neural networks.

[2]Levi D. McClenny, Ulisses M. Braga-Neto, Self-adaptive physics-informed neural networks, Journal of Computational Physics, Volume 474, (2023).

[3]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,Volume 378(2019)(P 686-707).