Fluid-Structure Interaction Analysis of Pulsatile Flow in Arterial Aneurysms with Physics-Informed Neural Networks and Computational Fluid Dynamics

Michael Ajao-Olarinoye[1]

[1]Center for computational science and Mathematical modelling

olarinoyem@coventry.ac.uk

1 Physics-Informed Neural Networks

1.1 Motivation and Overview

While high-fidelity Computational Fluid Dynamics (CFD) simulations in Section 2.4 provide accurate hemodynamic insights and remain a cornerstone for modelling fluid dynamics in complex vascular geometries, they are computationally intensive and often require significant domain expertise. This is especially true when investigating patient-specific geometries or performing multiple parameter studies, where each simulation run can be time-consuming and resource-intensive.

First introduced by **?**], Physics-Informed Neural Networks (PINNs) have emerged as a data-efficient alternative that merges the strengths of deep learning and classical physics-based modelling. In contrast to purely data-driven neural networks, PINNs embed the governing partial differential equations (PDEs)—for instance, continuity and momentum conservation from Section 2.2—and physical boundary conditions directly into the loss function. This approach enforces a fundamental level of physical consistency at every training iteration, meaning that rather than learning solely from black-box data, the network is "informed" by the equations that describe fluid motion. Consequently, PINNs often require fewer labelled data points, since the PDE residuals guide the network towards physically plausible solutions in data-scarce regions. Furthermore, the built-in PDE constraints can mitigate common neural network training pitfalls, such as overfitting or predicting unphysical flow fields.

In this work, we harness PINNs in tandem with traditional CFD for two primary objectives:

1. *Predict Flow Variables in Aortic Models.* PINNs are used to estimate pressure, velocity, and wall shear stress (WSS) within the same aortic geometries introduced in Section 2.1. These models include both healthy and Marfan Syndrome-affected aortas, enabling a direct comparison of hemodynamic features, such as local velocity patterns and shear forces on the vessel walls.

2. *Validate Against CFD Benchmarks.* By juxtaposing PINN predictions with high-fidelity CFD simulations, we assess how accurately the PINNs capture critical flow phenomena—particularly in aneurysmal segments, where accurate WSS quantification is essential for understanding disease progression in Marfan Syndrome.

1.2 Governing Equations for PINNs

We consider the unsteady, incompressible Navier–Stokes equations:

$$\nabla \cdot \mathbf{u} = 0, \quad \rho \frac{\partial \mathbf{u}}{\partial t} + \rho(\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \mu \nabla^2 \mathbf{u}, \tag{1}$$

where $\mathbf{u} = (u, v, w)$ is the velocity, p is the pressure, ρ is the density, and μ is the dynamic viscosity. The WSS components τ_x, τ_y, τ_z are derived from the velocity gradients at the vessel wall. These same equations form part of the PINN loss function, ensuring that the predicted solutions align with the underlying physics governing pulsatile blood flow.

1.3 PINN Architecture and Training

1.3.1 Network Configuration

We train seven separate PINNs: one each for pressure p, velocity components (u, v, w), and the three WSS components (τ_x, τ_y, τ_z). Each network is a feed-forward, fully connected architecture with 10 hidden layers of 64 neurons per layer, employing the Swish activation function to promote smooth gradients.

1.3.2 Input and Output

- **Inputs:** 4D coordinates $\{x, y, z, t\}$, corresponding to spatial dimensions and time.

- **Outputs:** One scalar field per network (*e.g.*, p for the pressure PINN, τ_x for the x-component of WSS).

1.3.3 Loss Functions

The total loss function employed during the training of the PINNs comprises multiple components, each targeting a specific aspect of the model's performance. These loss components are:

1. **Physics Loss ($\mathcal{L}_{\text{physics}}$):** This loss term enforces the satisfaction of the governing PDEs by minimising the residuals of the Navier–Stokes and continuity equations. Mathematically, it is expressed as:

$$\mathcal{L}_{\text{physics}} = \frac{1}{N_{\text{phys}}} \sum_{j=1}^{N_{\text{phys}}} \left(R_{u,j}^2 + R_{v,j}^2 + R_{w,j}^2 + R_{\text{cont},j}^2 \right), \tag{2}$$

   where $R_{u,j}$, $R_{v,j}$, $R_{w,j}$ are the residuals of the momentum equations in the x, y, z directions, respectively, and $R_{\text{cont},j}$ is the residual of the continuity equation at the j-th collocation point.

2. **Boundary Loss ($\mathcal{L}_{\text{boundary}}$):** This loss term enforces the no-slip boundary conditions by ensuring that the velocity components at the vessel walls are zero. It is defined as:

$$\mathcal{L}_{\text{boundary}} = \frac{1}{N_{\text{boundary}}} \sum_{k=1}^{N_{\text{boundary}}} \left(u_k^2 + v_k^2 + w_k^2 \right), \tag{3}$$

   where u_k, v_k, w_k are the predicted velocity components at the k-th boundary point.

3. **Data Loss ($\mathcal{L}_{\text{data}}$):** This supervised loss term quantifies the discrepancy between the PINN predictions and the CFD-generated ground-truth data. It is expressed as:

$$\mathcal{L}_{\text{data}} = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \Big((p_i - p_i^{\text{CFD}})^2 + (u_i - u_i^{\text{CFD}})^2 + (v_i - v_i^{\text{CFD}})^2 + (w_i - w_i^{\text{CFD}})^2 + (\tau_{x,i} - \tau_{x,i}^{\text{CFD}}) \tag{4}$$

   where p_i, u_i, v_i, w_i, $\tau_{x,i}$, $\tau_{y,i}$, $\tau_{z,i}$ are the PINN-predicted pressure, velocity components, and WSS components at the i-th data point, and p_i^{CFD}, u_i^{CFD}, v_i^{CFD}, w_i^{CFD}, $\tau_{x,i}^{\text{CFD}}$, $\tau_{y,i}^{\text{CFD}}$, $\tau_{z,i}^{\text{CFD}}$ are the corresponding CFD values.

4. **Inlet Loss ($\mathcal{L}_{\text{inlet}}$):** This loss term enforces the prescribed pulsatile inflow velocity profile at the inlet. Assuming a sinusoidal velocity profile, it is formulated as:

$$\mathcal{L}_{\text{inlet}} = \frac{1}{N_{\text{inlet}}} \sum_{m=1}^{N_{\text{inlet}}} \left((u_m - u_{\text{inlet}}(t_m))^2 + v_m^2 + w_m^2 \right), \tag{5}$$

where $u_{\text{inlet}}(t_m)$ is the prescribed inlet velocity at time t_m, and u_m, v_m, w_m are the predicted velocity components at the m-th inlet point.

The overall loss function utilised for training is a weighted sum of these components, where the weights are self-adaptive and learnable during training. The total loss is defined as:

$$\mathcal{L}_{\text{total}} = \exp(\log \lambda_{\text{physics}})\, \mathcal{L}_{\text{physics}} + \exp(\log \lambda_{\text{boundary}})\, \mathcal{L}_{\text{boundary}} + \exp(\log \lambda_{\text{data}})\, \mathcal{L}_{\text{data}} + \exp(\log \lambda_{\text{inle} \tag{6}$$

where $\log \lambda_i$ are learnable parameters corresponding to each loss component. During backpropagation, these parameters adjust to balance the influence of each loss term dynamically, enhancing training stability and performance.

1.3.4 Self-Adaptive Loss Weighting

Self-adaptive loss weighting obviates the need to manually tune the multipliers λ_{physics}, $\lambda_{\text{boundary}}$, λ_{data}, and λ_{inlet}, which can be particularly challenging in multi-physics scenarios involving PDE constraints, boundary conditions, and partial data supervision. By parameterising the weights as $\exp(\log \lambda_i)$, the model ensures that the weights remain positive throughout training, and the network can autonomously adjust the relative importance of each loss component based on the training dynamics.

1.3.5 Optimisation Strategy

The optimisation strategy encompasses several critical components to ensure effective and stable training of the PINNs:

- **Optimizer:** We employ the AdamW optimiser [**?**] with an initial learning rate of 1×10^{-4} and momentum parameters $\beta = (0.9, 0.999)$. The weight decay is set to 1×10^{-4} to prevent overfitting.

- **Learning Rate Scheduler:** A StepLR scheduler reduces the learning rate by a factor of $\gamma = 0.9$ every 200 epochs, facilitating finer convergence as training progresses.
- **Mixed Precision Training:** Automatic Mixed Precision (AMP) is utilised to leverage GPU efficiency, enabling faster computations without compromising numerical stability.
- **Early Stopping:** An early stopping mechanism monitors the validation loss for improvements, halting training after 5 consecutive epochs without a decrease in validation loss to avoid overfitting.
- **Gradient Clipping:** To prevent exploding gradients, the gradients are clipped to a maximum norm of 1.0, ensuring training stability.

1.4 Loss Function Implementation

The loss function implementation is central to the effectiveness of the PINNs. Each loss component targets specific physical and data-driven constraints, ensuring that the neural networks produce solutions that are both physically plausible and consistent with high-fidelity CFD data. Below, we detail the mathematical formulation and implementation of each loss component.

1.4.1 Physics Loss ($\mathcal{L}_{\text{physics}}$)

The Physics Loss enforces the satisfaction of the Navier–Stokes and continuity equations across collocation points. For each collocation point $\mathbf{x}_j^{\text{phys}} = (x_j, y_j, z_j, t_j)$, the residuals of the governing equations are computed using automatic differentiation. These residuals quantify the deviation of the PINN predictions from the exact PDEs.

Mathematically, the residuals are defined as:

$$R_{u,j} = \rho \left(\frac{\partial u_j}{\partial t} + u_j \frac{\partial u_j}{\partial x} + v_j \frac{\partial u_j}{\partial y} + w_j \frac{\partial u_j}{\partial z} \right) + \frac{\partial p_j}{\partial x} - \mu \left(\frac{\partial^2 u_j}{\partial x^2} + \frac{\partial^2 u_j}{\partial y^2} + \frac{\partial^2 u_j}{\partial z^2} \right), \tag{7}$$

$$R_{v,j} = \rho \left(\frac{\partial v_j}{\partial t} + u_j \frac{\partial v_j}{\partial x} + v_j \frac{\partial v_j}{\partial y} + w_j \frac{\partial v_j}{\partial z} \right) + \frac{\partial p_j}{\partial y} - \mu \left(\frac{\partial^2 v_j}{\partial x^2} + \frac{\partial^2 v_j}{\partial y^2} + \frac{\partial^2 v_j}{\partial z^2} \right), \tag{8}$$

$$R_{w,j} = \rho \left(\frac{\partial w_j}{\partial t} + u_j \frac{\partial w_j}{\partial x} + v_j \frac{\partial w_j}{\partial y} + w_j \frac{\partial w_j}{\partial z} \right) + \frac{\partial p_j}{\partial z} - \mu \left(\frac{\partial^2 w_j}{\partial x^2} + \frac{\partial^2 w_j}{\partial y^2} + \frac{\partial^2 w_j}{\partial z^2} \right), \tag{9}$$

$$R_{\text{cont},j} = \frac{\partial u_j}{\partial x} + \frac{\partial v_j}{\partial y} + \frac{\partial w_j}{\partial z}. \tag{10}$$

The Physics Loss is then formulated as the Mean Squared Error (MSE) of these residuals across all collocation points:

$$\mathcal{L}_{\text{physics}} = \frac{1}{N_{\text{phys}}} \sum_{j=1}^{N_{\text{phys}}} \left(R_{u,j}^2 + R_{v,j}^2 + R_{w,j}^2 + R_{\text{cont},j}^2 \right), \tag{11}$$

where N_{phys} is the total number of collocation points.

1.4.2 Boundary Loss ($\mathcal{L}_{\text{boundary}}$)

The Boundary Loss enforces the no-slip boundary conditions at the vessel walls, ensuring that the velocity components parallel and perpendicular to the wall are zero. For each boundary point $\mathbf{x}_k^{\text{boundary}} = (x_k, y_k, z_k, t_k)$, the boundary conditions are applied as:

$$u_k = 0, \tag{12}$$

$$v_k = 0, \tag{13}$$

$$w_k = 0. \tag{14}$$

The Boundary Loss is formulated as the MSE of the predicted velocities at these points:

$$\mathcal{L}_{\text{boundary}} = \frac{1}{N_{\text{boundary}}} \sum_{k=1}^{N_{\text{boundary}}} \left(u_k^2 + v_k^2 + w_k^2 \right), \tag{15}$$

where N_{boundary} is the number of boundary points.

1.4.3 Data Loss ($\mathcal{L}_{\text{data}}$)

The Data Loss quantifies the discrepancy between the PINN predictions and the CFD-generated ground-truth data. This supervised loss ensures that the neural networks align with high-fidelity simulations where data is available.

For each data point $\mathbf{x}_i^{\text{data}} = (x_i, y_i, z_i, t_i)$ with corresponding CFD data $\mathbf{q}_i^{\text{CFD}} = (p_i^{\text{CFD}}, u_i^{\text{CFD}}, v_i^{\text{CFD}}, w_i^{\text{CFD}}, \tau_{x,i}^{\text{CFD}}, \tau_{y,i}^{\text{CFD}}, \tau_{z,i}^{\text{CFD}})$, the Data Loss is defined as:

$$\mathcal{L}_{\text{data}} = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \Big[(p_i - p_i^{\text{CFD}})^2 + (u_i - u_i^{\text{CFD}})^2 + (v_i - v_i^{\text{CFD}})^2 + (w_i - w_i^{\text{CFD}})^2 + (\tau_{x,i} - \tau_{x,i}^{\text{CFD}})^2 + (\tau \tag{16}$$

where N_{data} is the number of data points.

1.4.4 Inlet Loss ($\mathcal{L}_{\text{inlet}}$)

The Inlet Loss enforces the prescribed pulsatile inflow velocity profile at the inlet of the aortic models. Assuming a sinusoidal velocity profile based on physiological blood flow characteristics, the Inlet Loss is formulated as follows.

For each inlet point $\mathbf{x}_m^{\text{inlet}} = (x_m, y_m, z_m, t_m)$ with prescribed inlet velocity $u_{\text{inlet}}(t_m)$, the Inlet Loss is defined as:

$$\mathcal{L}_{\text{inlet}} = \frac{1}{N_{\text{inlet}}} \sum_{m=1}^{N_{\text{inlet}}} \Big[(u_m - u_{\text{inlet}}(t_m))^2 + v_m^2 + w_m^2 \Big], \tag{17}$$

where N_{inlet} is the number of inlet points, and $u_{\text{inlet}}(t_m)$ represents the prescribed pulsatile inlet velocity at time t_m. The velocity components v_m and w_m are also penalised to enforce a predominantly unidirectional flow at the inlet.

1.4.5 Self-Adaptive Loss Weighting

A key feature of our PINNs is the utilisation of self-adaptive weights to balance the different components of the loss function. Specifically, the total loss $\mathcal{L}_{\text{total}}$ is expressed as:

$$\mathcal{L}_{\text{total}} = \exp(\log \lambda_{\text{physics}})\, \mathcal{L}_{\text{physics}} + \exp(\log \lambda_{\text{boundary}})\, \mathcal{L}_{\text{boundary}} + \exp(\log \lambda_{\text{data}})\, \mathcal{L}_{\text{data}} + \exp(\log \lambda_{\text{inle} \tag{18}$$

where $\log \lambda_i$ are learnable parameters that self-adapt to balance each term. This ensures that one loss component does not dominate the training, improving stability and convergence across different flow regimes. By parameterising the weights as $\exp(\log \lambda_i)$, we guarantee that the weights remain positive during training.

1.4.6 Algorithm: Training Procedure

The training procedure for the PINNs is encapsulated in Algorithm **??**. This algorithm outlines the step-by-step workflow, including data preparation, forward and backward passes, loss computations, self-adaptive weighting, optimisation, validation, and early stopping mechanisms.

Algorithm 1 Training Procedure for Transient Physics-Informed Neural Networks (PINNs)

Require:
- **PINN Models:** $\{\mathcal{N}_p, \mathcal{N}_u, \mathcal{N}_v, \mathcal{N}_w, \mathcal{N}_{\tau_x}, \mathcal{N}_{\tau_y}, \mathcal{N}_{\tau_z}\}$ with parameters $\theta_p, \theta_u, \theta_v, \theta_w, \theta_{\tau_x}, \theta_{\tau_y}, \theta_{\tau_z}$.
- **Training Data:** $\mathcal{D}_{\text{train}} = \{\mathbf{x}_i^{\text{data}}, \mathbf{q}_i^{\text{CFD}}\}$ containing $(p, u, v, w, \tau_x, \tau_y, \tau_z)$ from CFD for supervised loss.
- **Collocation Points:** $\mathcal{D}_{\text{phys}} = \{\mathbf{x}_j^{\text{phys}}\}$ for enforcing PDE constraints.
- **Hyperparameters:** total epochs T, batch size B, learning rate η, patience P, etc.

Ensure:
- **Trained PINN Parameters:** $\theta_p^*, \theta_u^*, \theta_v^*, \theta_w^*, \theta_{\tau_x}^*, \theta_{\tau_y}^*, \theta_{\tau_z}^*$.
- **Loss History:** Recorded losses for analysis.

Initialise the optimiser (e.g. AdamW) with all PINN parameters.
Initialise learning rate scheduler (e.g. StepLR).
Initialise early stopping mechanism with patience P.
Initialise GradScaler for mixed precision training.
for epoch $t = 1$ to T **do**
  Shuffle $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{phys}}$.
  Partition $\mathcal{D}_{\text{train}}$ into mini-batches $\mathcal{B}_{\text{data}}$, and $\mathcal{D}_{\text{phys}}$ into mini-batches $\mathcal{B}_{\text{phys}}$.
  for each mini-batch $\mathcal{B}_{\text{data}}^k$ and $\mathcal{B}_{\text{phys}}^k$ **do**
    Forward Pass:
    Compute $\{p_{\text{pred}}, u_{\text{pred}}, v_{\text{pred}}, w_{\text{pred}}, \tau_{x,\text{pred}}, \tau_{y,\text{pred}}, \tau_{z,\text{pred}}\}$ for $\mathcal{B}_{\text{data}}^k$ via each respective PINN.
    Compute Residuals:
    Calculate physics residuals $R_{u,j}$, $R_{v,j}$, $R_{w,j}$, $R_{\text{cont},j}$ at collocation points in $\mathcal{B}_{\text{phys}}^k$ using automatic differentiation.
    Loss Computations:
    $\mathcal{L}_{\text{physics}} \leftarrow \text{MSE}(R_{u,j}, 0) + \text{MSE}(R_{v,j}, 0) + \text{MSE}(R_{w,j}, 0) + \text{MSE}(R_{\text{cont},j}, 0)$
    $\mathcal{L}_{\text{boundary}} \leftarrow \text{MSE}(u_k, 0) + \text{MSE}(v_k, 0) + \text{MSE}(w_k, 0)$
    $\mathcal{L}_{\text{data}} \leftarrow \text{MSE}(p_i, p_i^{\text{CFD}}) + \text{MSE}(u_i, u_i^{\text{CFD}}) + \text{MSE}(v_i, v_i^{\text{CFD}}) + \text{MSE}(w_i, w_i^{\text{CFD}}) + \text{MSE}(\tau_{x,i}, \tau_{x,i}^{\text{CFD}}) + \text{MSE}(\tau_{y,i}, \tau_{y,i}^{\text{CFD}}) + \text{MSE}(\tau_{z,i}, \tau_{z,i}^{\text{CFD}})$
    $\mathcal{L}_{\text{inlet}} \leftarrow \text{MSE}(u_m, u_{\text{inlet}}(t_m)) + \text{MSE}(v_m, 0) + \text{MSE}(w_m, 0)$
    Self-Adaptive Weighting:
    $\lambda_{\text{physics}} \leftarrow \exp(\log \lambda_{\text{physics}})$
    $\lambda_{\text{boundary}} \leftarrow \exp(\log \lambda_{\text{boundary}})$
    $\lambda_{\text{data}} \leftarrow \exp(\log \lambda_{\text{data}})$
    $\lambda_{\text{inlet}} \leftarrow \exp(\log \lambda_{\text{inlet}})$
    Total Loss:
    $\mathcal{L}_{\text{total}} \leftarrow \lambda_{\text{physics}} \mathcal{L}_{\text{physics}} + \lambda_{\text{boundary}} \mathcal{L}_{\text{boundary}} + \lambda_{\text{data}} \mathcal{L}_{\text{data}} + \lambda_{\text{inlet}} \mathcal{L}_{\text{inlet}}$
    Backward Pass and Optimisation:
    optimizer.zero_grad()
    scaler.scale($\mathcal{L}_{\text{total}}$).*backward*()
    Gradient Clipping: Apply gradient clipping to cap gradients at norm 1.0.
    scaler.step(optimizer)
    scaler.update()
  end for
  Validation Phase:
  Compute validation loss $\mathcal{L}_{\text{val}}$ on the validation set.
  Early Stopping Check:
  if $\mathcal{L}_{\text{val}}$ has improved **then**
    Save current best model parameters $\theta_p^*, \theta_u^*, \theta_v^*, \theta_w^*, \theta_{\tau_x}^*, \theta_{\tau_y}^*, \theta_{\tau_z}^*$.

Explanation of Algorithm ??.

- **Initialisation:** Seven PINN models are instantiated for the flow variables of interest. An optimiser (e.g. AdamW) and learning rate scheduler (e.g. StepLR) are configured, alongside an early stopping mechanism and a GradScaler for mixed-precision training on GPU.
- **Epoch Loop:** For each epoch, the training dataset ($\mathcal{D}_{\text{train}}$) and the collocation points ($\mathcal{D}_{\text{phys}}$) are shuffled and batched. The algorithm then iterates over mini-batches for both data and physics-based enforcement.
- **Forward Pass:** For each mini-batch, predictions of pressure, velocity, and WSS are computed. Simultaneously, physics residuals (*i.e.*, partial derivatives enforcing PDE constraints) are calculated using automatic differentiation.
- **Loss Computations:** The loss components are assembled—*i.e.*, physics loss ($\mathcal{L}_{\text{phys}}$), boundary loss ($\mathcal{L}_{\text{boundary}}$), data loss ($\mathcal{L}_{\text{data}}$), and inlet loss ($\mathcal{L}_{\text{inlet}}$). Each term is scaled by an exponentiated, learnable weight λ_i, enabling the network to balance different objectives adaptively.
- **Backward Pass and Optimisation:** The total loss is backpropagated, with gradient clipping applied to ensure training stability, and the weights are updated. Mixed precision training is facilitated by the GradScaler, which dynamically adjusts scaling factors for numerical efficiency on modern GPUs.
- **Validation and Early Stopping:** After all mini-batches are processed, a validation set is used to monitor progress. If validation loss fails to improve for a specified patience period, training terminates early.
- **Learning Rate Adjustment and Logging:** A learning rate scheduler refines the step size over epochs, and key metrics, such as losses and adaptive weights, are logged to track training progress.

1.5 Relevance to Aneurysm Studies

By alleviating the need to run complete CFD simulations whenever boundary conditions or model geometries change, PINNs can serve as rapid surrogates for iterative

parameter sweeps, sensitivity analyses, or real-time clinical decision-making. This synergy between classical CFD and PINNs ensures that accurate fluid physics are retained, while the neural network structure offers improved computational scaling and the capacity to generalise to different flow conditions with minimal retraining overhead. Consequently, for the case of Marfan Syndrome aortic aneurysms, PINNs provide a promising pathway for expediting haemodynamic evaluations and exploring a range of inflow waveforms, wall properties, or geometric variations in a fraction of the time typically required by CFD alone.

Note: All codes and numerical experiments for the PINN-based aneurysm flow analyses were implemented in `Python` using `PyTorch`, with the self-adaptive weighting and physics-based PDE residuals integrated into the backpropagation routine. The specific hyperparameters and training protocol are as outlined in Algorithm **??** and Section 3.4, ensuring reproducibility and transparency in the results reported herein.

References

[] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.