Numerical Simulation of Blood flow in Arteries

S Soundarya IPHD20025

School of Mathematics, IISER TVM

Master's Project Presentation

May 2023

Aim

- Simulating blood flow in a curved domain through Finite Element Method(FEM) using DUNE[1] in two and three dimensions.
- Implementing a mixed-variable Physics Influenced Neural Networks(PINNs) formulation proposed in [2] for simulating blood flow in a circular arch.
- Qualitative comparison of the results obtained through PINNs and FEM.

Table of Contents

Cardiovascular system

Major circulation paths of Cardiovascular system:
(i) pulmonary circulation, which carries blood between the heart and lungs,
(ii) systemic circulation, which carries blood from the heart through the body and back again.
The systemic circulation begins in the left atrium.

- Receives oxygenated blood from the pulmonary veins.
- Blood moves from the left atrium to the left ventricle where it is pumped into the aorta.
- From aorta the blood travels into the capillaries and returns to the right atrium through the veins.

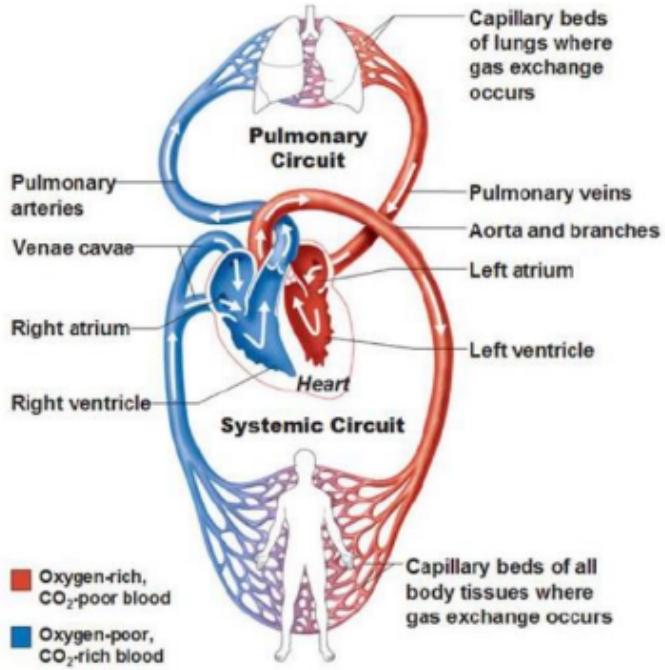


Figure 1: Cardiovascular System

Navier Stokes Equation using Finite Element Method

Assumptions on Blood:

- Blood is a Newtonian Fluid
- Blood is an incompressible fluid and a homogeneous fluid.

Let $\Omega \subset R^d$ be an open, bounded Lipschitz domain with boundary Γ.

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} - \nu \Delta \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p &= \mathbf{f} \quad \text{in } \Omega \\ \nabla \cdot \mathbf{u} &= 0 \quad \text{in } \Omega \\ \mathbf{u} &= \mathbf{g} \quad \text{on } \Gamma_D \\ \nu \frac{\partial \mathbf{u}}{\partial n} - p\mathbf{n} &= \mathbf{s} \quad \text{on } \Gamma_D \end{aligned} \tag{1}$$

where $\nu := \frac{\mu}{\rho}$ is the kinematic viscosity, μ is the dynamic viscosity, ρ is the density of the fluid. $(\mathbf{u} \cdot \nabla)\mathbf{u}$ describes the process of convective transport $(\nu \Delta \mathbf{u})$ molecular diffusion.

Some function spaces

- Let $-\infty \leq a < b \leq \infty$ and X be a Banach space normed by $||\cdot||_X$.

  $L^p(a,b;X) := \{f : (a,b) \longrightarrow X| \quad t \longrightarrow ||f(t)||_X \text{ is strongly measurable } \}$

  $||f||_{L^P(a,b;X)} = (\int_\Omega ||f||_X^p)^{\frac{1}{p}} < \infty,\ 1 \leq p < \infty$

  $||f||_{L^\infty(a,b;X)} = \sup_{t\in(a,b)} \text{ess } ||f(t)||_X < \infty, \text{ if } p = \infty.$

- $\mathbf{H}^1_{\mathbf{g}} := \{\mathbf{u} \in \mathbf{H}^1 | \mathbf{u} = \mathbf{g} \text{ on } \partial\Omega_D\}$

- $\mathbf{H}^1_{\mathbf{0}} := \{\mathbf{u} \in \mathbf{H}^1 | \mathbf{u} = \mathbf{0} \text{ on } \partial\Omega_D\}$

- $\mathbf{v} \in L^p(\Omega)$ is called to be weakly divergence-free if $\int_\Omega \nabla\psi \cdot \mathbf{v} = \mathbf{0} \quad \forall\psi \in D(\Omega)$.

Weak Formulation of Navier Stokes Equation

A weak formulation of the Navier Stokes Equation can be derived by multiplying by test functions $(\mathbf{v}, q)$ from suitable function spaces and integrating it over Ω

$$\int_\Omega \frac{\partial \mathbf{u}}{\partial t} \cdot \mathbf{v} d\Omega - \nu \int_\Omega \Delta \mathbf{u} \cdot \mathbf{v} d\Omega + \int_\Omega (\mathbf{u} \cdot \nabla)\mathbf{u} \cdot \mathbf{v} d\Omega + \int_\Omega \nabla p \cdot \mathbf{v} d\Omega = \int_\Omega \mathbf{f} \cdot \mathbf{v} d\Omega$$

.

Using Green's theoram, we obtain

$$\int_\Omega \frac{\partial \mathbf{u}}{\partial t} \cdot \mathbf{v} + \int_\Omega \nu(\nabla \mathbf{u} : \nabla \mathbf{v}) + \int_\Omega (\mathbf{u} \cdot \nabla)\mathbf{u} \cdot \mathbf{v} - \int_\Omega p \nabla \cdot \mathbf{v} = \int_\Omega \mathbf{f} \cdot \mathbf{v} + \int_{d\Omega} (\nu \frac{\partial \mathbf{u}}{\partial n} - p\mathbf{n}) \cdot \mathbf{v}$$

$$\int_\Omega q \nabla \cdot \mathbf{u} = 0 \qquad \forall \quad (\mathbf{v}, q)$$

$\nabla \mathbf{u} : \nabla \mathbf{v}$ represents the componentwise scalar product $\nabla \mathbf{u}_x \cdot \nabla \mathbf{v}_x + \nabla \mathbf{u}_y \cdot \nabla \mathbf{v}_y$ for two dimensions.

Find $\mathbf{u} \in \mathbf{H}^1_{\mathbf{g}}$ and $p \in L^2(\Omega)$ such that

$$\int_\Omega \frac{\partial \mathbf{u}}{\partial t} \cdot \mathbf{v} + \int_\Omega \nu(\nabla \mathbf{u} : \nabla \mathbf{v}) + \int_\Omega (\mathbf{u} \cdot \nabla)\mathbf{u} \cdot \mathbf{v} - \int_\Omega p \nabla \cdot \mathbf{v} = \int_\Omega \mathbf{f} \cdot \mathbf{v} + \int_{d\Omega} \mathbf{s} \cdot \mathbf{v}$$

$$\int_\Omega q \nabla \cdot \mathbf{u} = 0$$

for all $\mathbf{v} \in \mathbf{H}^1_{\mathbf{0}}$ and $q \in L^2(\Omega)$.

Discretization of Navier Stokes

Space Discretization:

Define the following finite dimensional spaces for an integer $r \geq 0$

$$\mathbf{V_h} := \{\mathbf{v} \in \mathbf{V} \quad : \quad \mathbf{v}|_K \in \mathbf{P_{r+1}}(K) \quad \forall K \in T_h\} \subset \mathbf{V} = \mathbf{H_0^1}$$

$$\mathbf{U_h} := \{\mathbf{u} = \mathbf{u}_g + \mathbf{v} \quad : \quad \mathbf{u}_g = \mathbf{g} \text{ on } \Gamma_D \quad \mathbf{v} \in \mathbf{V_h} \quad \forall K \in T_h\} \subset \mathbf{U} = \mathbf{H_g^1}$$

$$M_h := \{p \in M \quad | \quad p|_K \in P_r(K) \quad \forall K \in T_h\} \subset M = L^2(\Omega)$$

Find $\mathbf{u}_h \in \mathbf{U}_h$ and $p \in M_h$ such that

$$\int_\Omega \frac{\partial \mathbf{u}_h}{\partial t} \cdot \mathbf{v}_h + \int_\Omega \nu(\nabla \mathbf{u}_h : \nabla \mathbf{v}_h) + \int_\Omega (\mathbf{u}_h \cdot \nabla)\mathbf{u}_h \cdot \mathbf{v}_h - \int_\Omega p_h \nabla \cdot \mathbf{v}_h = \int_\Omega \mathbf{f} \cdot \mathbf{v}_h + \int_{d\Omega} \mathbf{s} \cdot \mathbf{v}_h$$

$$\int_\Omega q_h \nabla \cdot \mathbf{u}_h = 0$$

for all $\mathbf{v}_h \in \mathbf{V}_h$ and $q_h \in M_h$.

Lagrange Basis functions

Introduce a set of vector-valued (velocity) basis functions $\{\phi_j\}$. In two dimensions

$$\{\phi_1, \phi_2, \cdots, \phi_{2n}\} := \{(\phi_1, 0)^t, \cdots, (\phi_n, 0)^t, (0, \phi_1)^t, \cdots, (0, \phi_n)^t\}$$

Each basis function is of Lagrangian type: each basis function ϕ_j has a node $x_j \in \Omega$

$$\phi_j(x_j) = 1, \quad \phi_j(x_i) = 0 \quad \text{for all nodes } x_i \neq x_j$$

Similarly introduce a set of scalar(pressure) basis function $\{\psi_k\}$. To identify the corresponding linear algebra problem, set $(\mathbf{u}_h, p_h)$ as

$$\mathbf{u}_h = \sum_{j=1}^{n_u} \mathbf{u}_j \phi_j + \sum_{j=n_u+1}^{n_u+n_\delta} \mathbf{u}_j \phi_j \qquad p_h = \sum_{k=1}^{n_p} p_k \psi_k$$

with $\sum_{j=1}^{n_u} \mathbf{u}_j \phi_j \in V_h$.

Fix the coefficients $u_j : j = n_u + 1, \ldots, n_u + n_\partial$ so that the second term interpolates the boundary data on $\partial\Omega_D$, that is, for $x_j \in \partial\Omega_D$, $\mathbf{u}_h(x_j) = \mathbf{u}_j = \mathbf{g}_j$.

Time Discretization

Let $0 = t_0 < t_1 < \cdots < t_N = I$ be a uniform decomposition of the considered time interval $[0, I]$ and $\delta_t = t^n - t^{n-1}, 1 \leq n \leq N$, be the time step size. Using the Taylor expansion at $t^* = \theta t^n + (1-\theta)t^{n-1}$ for $0 \leq \theta \leq 1$,

$$\frac{u(t^n) - u(t^{n-1})}{\delta_t} = u'(t^*) + \frac{(1-2\theta)}{2}\delta_t u''(t^*) + O(\delta_t^2)$$

If $t^* \in [t^{n-1}, t^n]$, using the Taylor series expansion above we get,

$$\mathbf{u}_h(t^*) = \theta \mathbf{u}_h(t^n) + (1-\theta)\mathbf{u}_h(t^{n-1})$$

$$p_h(t^*) = \theta p_h(t^n) + (1-\theta)p_h(t^{n-1})$$

$$\mathbf{f}_h(t^*) = \mathbf{f}(\theta t^n + (1-\theta)t^{n-1})$$

The system of equations obtained using the symmetric tensor $D(\mathbf{u}) = \frac{\nabla \mathbf{u} + \nabla \mathbf{u}^T}{2}$ is

$$\begin{cases} (\mathbf{u}^{n+1}, \mathbf{v}_h) + \theta \Delta t [2\nu D(\mathbf{u}^{n+1}), \nabla \mathbf{v}_h) + ((\mathbf{u}^{n+1} \cdot \nabla)\mathbf{u}^{n+1}, \mathbf{v}_h)] - \Delta t \theta (\nabla \cdot \mathbf{v}_h, p^{n+1}) \\ = (\mathbf{u}^n, \mathbf{v}_h) - (1-\theta)\Delta t [2\nu (D(\mathbf{u}^n), \nabla \mathbf{v}_h) + ((\mathbf{u}^n \cdot \nabla)\mathbf{u}^n, \mathbf{v}_h)] + \Delta t (1-\theta)(\nabla \cdot \mathbf{v}_h, p^n) \\ +(1-\theta)\Delta t < \mathbf{f^n}, \mathbf{v}_h > + \theta \Delta t < \mathbf{f^{n+1}}, \mathbf{v}_h > \\ \\ (\nabla \cdot \mathbf{u}^{n+1}, q_h) = 0 \end{cases}$$

for all $(\mathbf{v}_h, q_h) \in V_h \times M_h$

Crank Nicholson Scheme

Setting $\theta = 0.5$, we get the Crank Nicholson scheme which is second order accurate in time. The matrix system is non-linear. In two dimensions, we get,

$$\begin{pmatrix} M+\nu A+\nu S_{ax} & \nu S_{ay} & B_x^T \\ \nu S_{bx} & M+\nu A+\nu S_{by} & B_y^T \\ B_x & B_y & 0 \end{pmatrix} \begin{pmatrix} u_{x,k+1}^{n+1} \\ u_{y,k+1}^{n+1} \\ p_{k+1}^{n+1} \end{pmatrix} + \begin{pmatrix} N_x \\ N_y \\ 0 \end{pmatrix} = \begin{pmatrix} h_x(u_x^n, p^n, f_x^{n,n+1}) \\ h_y(u_y^n, p^n, f_y^{n,n+1}) \\ 0 \end{pmatrix}$$

where $M, A, S_{ax}, S_{ay}, S_{bx}, S_{by} \in R^{n_u \times n_u}$, $B_x, B_y \in R^{np \times n_u}$ and $N_x, N_y \in R^{n_u \times 1}$ and

$$[m]_{ij} = \frac{2}{\Delta t}\int_\Omega \phi_i\phi_j \quad [a]_{ij} = \int_\Omega \nabla\phi_i\nabla\phi_j \quad [s_{ax}]_{ij} = \int_\Omega \frac{\partial\phi_j}{\partial x}\frac{\partial\phi_i}{\partial x} \quad [b_x]_{ij} = -\int_\Omega \psi_i\frac{\partial\phi_j}{\partial x}$$

$$[s_{ay}]_{ij} = \int_\Omega \frac{\partial\phi_j}{\partial y}\frac{\partial\phi_i}{\partial x} \quad [s_{bx}]_{ij} = \int_\Omega \frac{\partial\phi_j}{\partial x}\frac{\partial\phi_i}{\partial y} \quad [s_{by}]_{ij} = \int_\Omega \frac{\partial\phi_j}{\partial y}\frac{\partial\phi_i}{\partial y} \quad [b_y]_{ij} = -\int_\Omega \psi_i\frac{\partial\phi_j}{\partial y}$$

$$[n_x]_i = \int_\Omega (\nabla u_{x,k+1}^{n+1})u_{x,k+1}^{n+1}\phi_i \quad [n_y]_i = \int_\Omega (\nabla u_{y,k+1}^{n+1})u_{y,k+1}^{n+1}\phi_i$$

Newton Method

The solution at the $(n+1)^{th}$ time step to be above system can be representd by the residual $R(\mathbf{u}^{n+1}, p^{n+1})$ as

$$R(\mathbf{u}^{n+1}, p^{n+1}) := [K][\mathbf{u}^{n+1}\ p^{n+1}]^T + N(\mathbf{u}^{n+1}, p^{n+1}) - H(\mathbf{u}^n, p^n, f^n) \tag{2}$$

The goal is to find a solution such that the above residual is minimized. Newton's method provides a way to compute a solution iteratively. The Jacobian matrix is computed by using numerical differentiation. Let $z = (\mathbf{u}^{n+1}, p^{n+1})$, then

$$[J]_{ij}(z) = \frac{\partial R_i}{\partial z_j}(\mathbf{u}^{n+1}, p^{n+1})$$

$$\frac{\partial R_i}{\partial z_j}(\mathbf{u}^{n+1}, p^{n+1}) = \frac{R_i(z_1, \ldots, z_j + \Delta z_j, \ldots, z_l)|_z - R_i(z_1, \ldots, z_j, \ldots, z_l)|_z}{\Delta z_j}$$

Algorithm 1 Newton's Iteration

1: Input tolerance tol, initial guess $(\mathbf{u}_k^n, p_k^n)$

2: $k = 0$

3: **while** True **do**

4: Compute $J(\mathbf{u}_k^{n+1}, p_k^{n+1})$ by numerical differentiation.

5: Solve for $(\mathbf{w}_k^{n+1}, l_k^{n+1})^T$ by using a suitable linear solver.

$$J(\mathbf{u}_k^{n+1}, p_k^{n+1}) \begin{pmatrix} \mathbf{w}_k^{n+1} \\ l_k^{n+1} \end{pmatrix} = R(\mathbf{u}_k^{n+1}, p_k^{n+1})$$

6: **if** $(||w_k^{n+1}||^2 + ||l_k^{n+1}||^2)^{\frac{1}{2}} < tol$ **then**

7: *break*

8: **end if**

9: Update

$$\begin{pmatrix} \mathbf{u}_{k+1}^{n+1} \\ p_{k+1}^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{u}_k^{n+1} \\ p_k^{n+1} \end{pmatrix} - \begin{pmatrix} \mathbf{w}_k^{n+1} \\ l_k^{n+1} \end{pmatrix}$$

10: $k = k + 1$

11: **end while**

Table of Contents

Fluid Flow and Geometry of the Domain

- Reynolds Number is a dimensionless number which depends upon the velocity of the fluid and is given as

$$Re = \frac{\rho|\mathbf{u}|d}{\mu} = \frac{\text{inertial forces}}{\text{viscous forces}}$$

- When Re is smaller, the viscous forces dominate over the inertial forces(the fluid is flowing slower) and the flow is laminar. When the Reynolds number exceeds a threshold value (called the critical Reynolds number), semi-developed turbulence occurs in the flow.
- When a fluid is moving in a curved path, each particle of the fluid is acted by a centrifugal force, which is directly proportional to the square of the body's velocity and inversely proportional to the radius of the curvature of its path.

Secondary Flow

- Velocity of the fluid particles close to the axis is higher compared to the one away from the edges, so, the fluid particles closer to the center of the pipe will be forced harder toward the outside of the bend and particels along wall will be forced inside.
- This outward and inward movement of the particles constitute the secondary flow. Eustice(1910) was able to prove this experimentally.
- Dean(1927) showed that there is only one governing parameter in the flow, the Dean number (measure of intensity of the secondary flow)

$$De = Re\sqrt{\delta} = \frac{\sqrt{(\text{inertial force})(\text{centrifugal force})}}{\text{viscous forces}}$$

where $\delta = \frac{a}{R}$ is the curvature, a : the radius of cross-section of the pipe, R : the radius of curvature at the pipe centerline.

Dean Number

- For $De < 40 \sim 60$ the flow is unidirectional. As the Dean number increases between $60 \sim 75$, some wavy perturbations occurs in the cross-section indicating some secondary flow. A secondary instability appears for $De > 75 \sim 200$, where the vortices present undulations, twisting, and eventually merging and pair splitting.
- Taylor(1929) and White(1929) showed experimentally that the Reynolds number at which laminar-turbulent transition occurs in helical pipes increases with the curvature. The flow in curved pipes is much more stable than that in a straight pipe, and the critical Reynolds number may be even twice (or more)

Test Examples

Example 1

- The computational domain is a semicircular pipe[3] with smooth disturbances in two dimension with cross sectional radius $a = 1.6cm$ and curvature radius $R = 2.9cm$.
- The kinematic viscosity $\nu = 0.4\ gs^{-1}cm^{-1}$.
- **Boundary Conditions:**

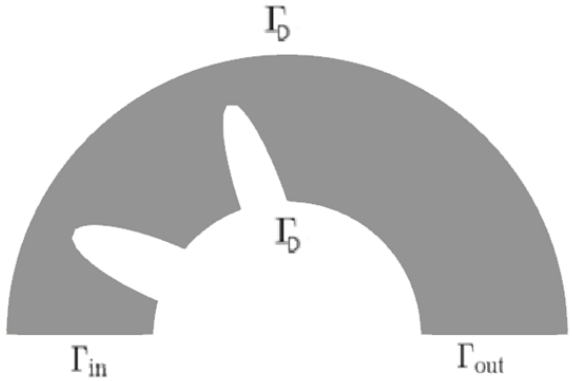
$$\mathbf{u} = 0 \qquad \text{on } \Gamma_D$$

$$\mathbf{u} = \begin{pmatrix} 0 \\ 3 \cdot x_1 \cdot \frac{(3.2-x_1)}{3.2 \cdot 3.2} \cdot (sin(\frac{\pi t}{6.0})) \end{pmatrix} \qquad \text{on } \Gamma_{in}$$

$$\nu \nabla \mathbf{u} \mathbf{n} - p\mathbf{n} = 0 \qquad \text{on } \Gamma_{out}$$

- **Initial conditions:** $(\mathbf{u}^0, p^0) = (\mathbf{0}, 0)$

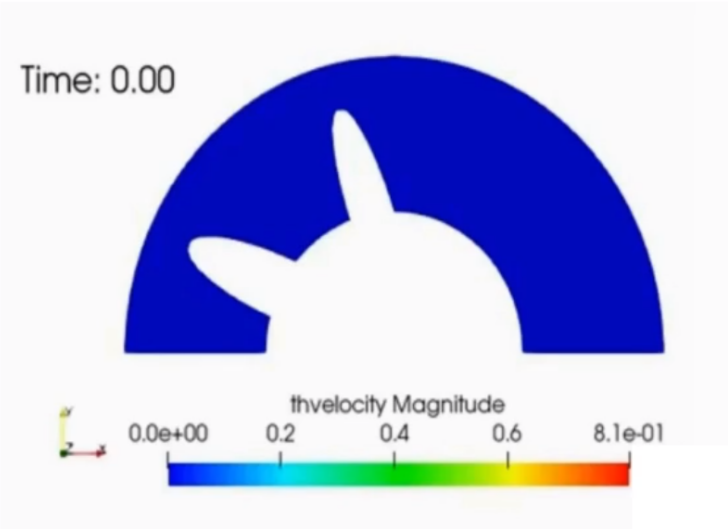
- Domain:



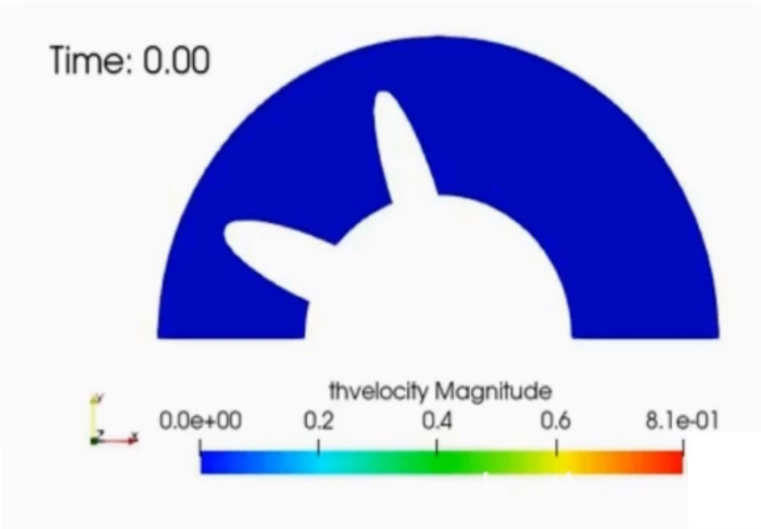
- Simulation carried out in DUNE[1] Framework. Geometry is created using GMSH[4] and Unstructured Grid is used for meshing. The velocity is of degree two and pressure is taken to be degree one.
- The number of elements in the mesh is 1936. The number of edges in the mesh is 3002. Total number of vertices in the mesh is 1067. The number of constrained degrees of freedom is 739.
- Crank Nicholson Scheme is used for discretization. The non linear solver used is Newton. Bi-Conjugate Gradient Stabilized with ILU Preconditioner used as linear solver.
- The simulation is done for $T = 6$ seconds with the time step $\Delta t = 0.01$ seconds. The CPU time taken for the simulation is 247.824 seconds.

Simulation

Viscosity=0.4 g/(cm s)



Viscosity=0.04 g/(cm s)



Observations:

- The Reynolds number of the flow is 22.875, 228.75 for viscosity 0.4 and 0.04 respectively.
- The Dean number for the viscosity $\nu = 0.4 gs^{-1} cm^{-1}$ is 18.894. The flow is completely unidirectional.
- The Dean number for viscosity $\nu = 0.04 gs^{-1} cm^{-1}$ is 188.94. A secondary instability appears at time $T = 4$ seconds there is a vortex formation leading to backflow instability.

Example 2

- The computational domain is a semi circular domain[5] in three dimension with cross sectional radius $R = 1.6\ cm$, curvature radius 2.9 cm and domain length $L = 12.2\ cm$. The dynamic viscosity $\mu = 0.4\ gcm^{-1}s^{-1}$, density $\rho = 1.06\ gcm^{-3}$, blood pressure drop $\Delta P = 1Pa$.
- **Boundary Conditions:**

$$\mathbf{u} = 0 \qquad \text{on } \Gamma_D$$

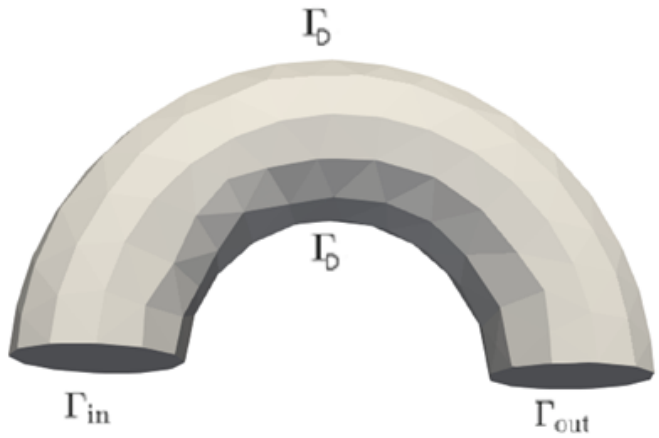
$$\mathbf{u} = \begin{pmatrix} 0 \\ (1 - (\frac{r}{R})^2) \cdot a \cdot Q \\ 0 \end{pmatrix} \qquad \text{on } \Gamma_{in}$$

$$\nu \nabla \mathbf{u} \mathbf{n} - p\mathbf{n} = 0, \quad p = 0 \qquad \text{on } \Gamma_{out}$$

where, $Q := \frac{\pi R^4 \Delta P}{8\mu L} = 5.2703\ cm^3 s^{-1}$ is the volumetric inflow rate and $a := \frac{2}{\pi R^2}$. $\mathbf{n}$ denotes the unit outer normal vector to the domain.

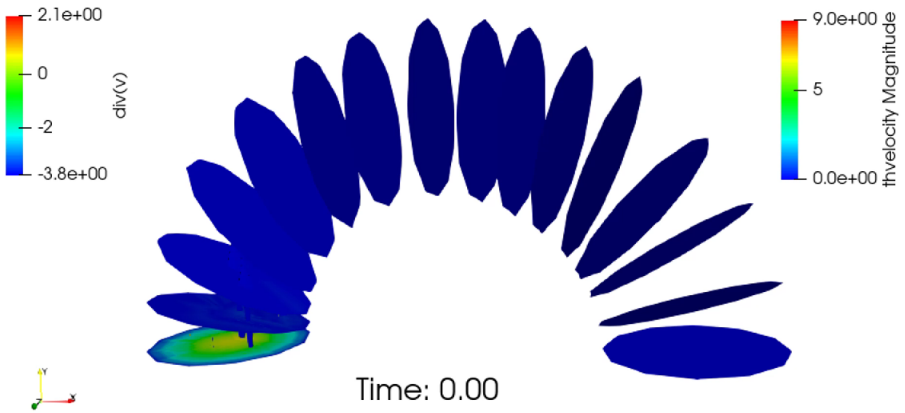
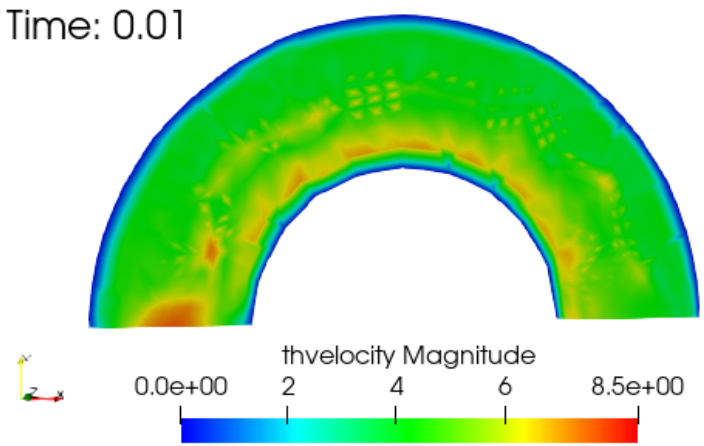
- **Initial conditions:** $(\mathbf{u}^0, p^0) = (\mathbf{0}, 0)$

- Domain:



- Simulation carried out in DUNE[1] Framework. Geometry is created using GMSH[4] and Unstructured Grid is used for meshing. The velocity is of degree three and pressure is taken to be degree two.
- The number of elements in the mesh is 1114. Total number of vertices in the mesh is 227. The number of constrained degrees of freedom is 5383.
- Crank Nicholson Scheme is used for discretization. The non linear solver used is Newton. Bi-Conjugate Gradient Stabilized with ILU Preconditioner is used as the linear solver. The simulation is done for $T = 2$ seconds with time $\Delta t = 0.01$ seconds. The CPU time taken for the simulation is 2739.57 seconds

Simulation

velocity	clip view
	

Observations:

- The Reynolds number of the flow is 161.
- The Dean number of the flow is 33.19 and the flow is laminar as expected.
- The maximal axial speed is pushed toward the outer bend of the pipe due to centrifugal force arising as a result of the curvature. This can be clearly seen if we compare the flow at $T = 0.40$ second and $T = 1.25$ second.

Table of Contents

Artificial Neural Networks

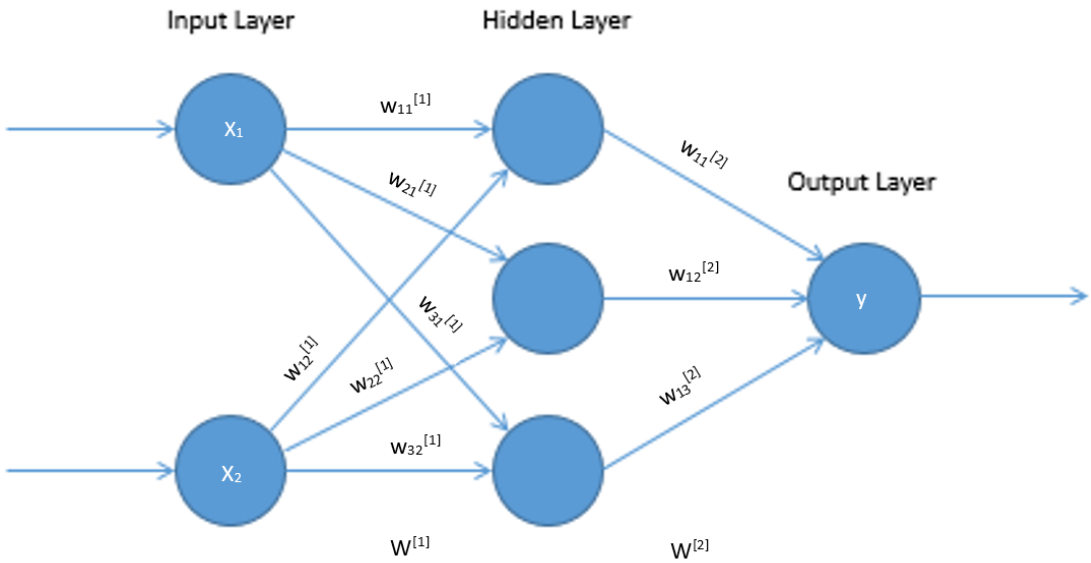
- The general idea of a Neural Network is to find a model that, based on a set of N samples,

$$D = [x_1, ..x_N], [y_1, .., y_N]$$

will approximate a unknown function f, with $f(x_i) = y_i$ as well as possible.

- Neural Network consists of Layers, Weights and Activation functions.
- Activation function are functions which model a threshold that decides whether the information that a neuron gets will be relevant for the further calculations or not.
- During the training phase, after processing (x_i, y_i) in a sample dataset D, the distance of the output that the network produces to the actual desired output y_i, is measured by some loss function.
- According to the loss, the weights of the transitions are modified to achieve a better result in the next iteration.

Example of a Neural Network



$w_{ij}^{[l]}$ is the weight associated with a node i of layer l.

The weight matrix associated with layer 1 is:

$$W^{[1]} = \begin{pmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \end{pmatrix}$$

Forward Pass

The output at layer l is given by the matrix $A^{[l]}$ as

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]} \qquad A^{[l]} = \sigma(W^{[l]}A^{[l-1]} + b^{[l]}) = \sigma(Z^{[l]})$$

with $A^{[0]}$ given by the input features. In this example, $A^{[0]} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$.

For the given network we have the output y to be

$$y = A^{[2]} = \sigma(W^{[2]}A^{[1]} + b^{[2]}) = \sigma(W^{[2]}(\sigma(W^{[1]}A^{[0]} + b^{[1]})) + b^{[2]})$$

This procedure of calculating the output y from the input training data is known as the forward pass.

The loss function is given by $C(w, b)$. The weights are then updated by using the backpropagation algorithm.

Backpropagation Algorithm

For a layer l let $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$, hence $A^{[l]} = \sigma(Z^{[l]})$. If l be the outermost layer, $\frac{\partial C}{\partial A^{[l]}}$ can be computed.

Algorithm 5 Backpropagation Algorithm

while $l \geq 1$ **do**

$\frac{\partial C}{\partial Z^{[l]}} = \sigma'(Z^{[l]})\frac{\partial C}{\partial A^{[l]}}$

$\frac{\partial C}{\partial W^{[l]}} = \frac{\partial C}{\partial Z^{[l]}}(A^{[l-1]})^T$

$\frac{\partial C}{\partial b^{[l]}} = \frac{\partial C}{\partial Z^{[l]}}$

$W^{[l]} = W^{[l]} - \alpha\frac{\partial C}{\partial W^{[l]}}$

$b^{[l]} = b^{[l]} - \alpha\frac{\partial C}{\partial b^{[l]}}$

$\frac{\partial C}{\partial A^{[l-1]}} = (W^{[l]})^T\frac{\partial C}{\partial Z[l]}$

$l = l - 1$

Auto Differention

Automatic differentiation evaluates expressions numerically at particular numeric values, rather than carrying out large symbolic computations.

Forward mode Auto differentiation

This evaluates a numerical derivative by performing elementary derivative operations concurrently with the operations of evaluating the function itself.

Forward mode Auto Differentiation

- $\hat{y}$ be the actual output and y be the output predicted by the neural network.
- $L = (y - \hat{y})^2$ and $w_{11}^{[2]}$ is updated as $w_{11}^{[2]} = w_{11}^{[2]} - \alpha \frac{\partial L}{\partial w_{11}^{[2]}}$
- $(a_1^{[1]}, a_2^{[1]}, a_3^{[1]})$ is the output of the first layer then the predicted output using the *tanh* activation function is given as

$$y = tanh(w_{11}^{[2]} \cdot a_1^{[1]} + w_{12}^{[2]} \cdot a_2^{[1]} + w_{13}^{[2]} \cdot a_3^{[1]})$$

$$L = (tanh(w_{11}^{[2]} \cdot a_1^{[1]} + w_{12}^{[2]} \cdot a_2^{[1]} + w_{13}^{[2]} \cdot a_3^{[1]}) - \hat{y})^2$$

- For computing the derivative of L with respect to $w_{11}^{[2]}$ associate each intermediate variable v_i a derivative

$$\dot{v}_i = \frac{\partial v_i}{\partial w_{11}^{[2]}}$$

$$L = (tanh(w_{11}^{[2]} \cdot a_1^{[1]} + w_{12}^{[2]} \cdot a_2^{[1]} + w_{13}^{[2]} \cdot a_3^{[1]}) - \hat{y})^2$$

Forward Primal Trace	Forward Tangent Trace
$v_{-2} = w_{11}^{[2]}$	$\dot{v}_{-2} = 1$
$v_{-1} = w_{12}^{[2]}$	$\dot{v}_{-1} = 0$
$v_0 = w_{13}^{[2]}$	$\dot{v}_0 = 0$
$v_1 = a_1^{[1]} \cdot v_{-2}$	$\dot{v}_1 = a_1^{[1]}$
$v_2 = a_2^{[1]} \cdot v_{-1}$	$\dot{v}_2 = 0$
$v_3 = a_3^{[1]} \cdot v_0$	$\dot{v}_3 = 0$
$v_4 = tanh(v_1 + v_2 + v_3)$	$\dot{v}_4 = sech^2(v_1 + v_2 + v_3)(\dot{v}_1 + \dot{v}_2 + \dot{v}_3) = sech^2(v_1 + v_2 + v_3) \cdot a_1^{[1]}$
$v_5 = v_4 - \hat{y}$	$\dot{v}_5 = \dot{v}_4$
$L = v_6 = (v_5)^2$	$\frac{\partial L}{\partial w_{11}^{[2]}} = \frac{\partial v_6}{\partial v_{-2}} = \dot{v}_6 = 2v_5 \cdot \dot{v}_5$

Figure 2: Finding $\frac{\partial L}{\partial w_{11}^{[2]}}$ by Automatic Differentiation

Table of Contents

- In a data-driven framework, the Deep learning model is constructed as a black-box to learn a surrogate mapping from the formatted input $x \in R^m$ to the output $y \in R^n$.
- Physics-Informed Neural Networks (PINNs) is a scientific machine learning technique used to solve problems involving Partial Differential Equations(PDE).
- PINNs approximate PDE solutions by training a neural network to minimize a loss function which includes terms reflecting the initial and boundary conditions and the PDE residual at selected points in the domain (called collocation point).
- PINN can be thought of as an unsupervised strategy that does not require labelled data, such as results from prior simulations or experiments.
- PINNs is essentially a mesh-free technique.

Navier Stokes Equation using PINNs

Navier stokes equation for an incompressible flow

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2 \mathbf{v}$$
$$\nabla \cdot \mathbf{v} = 0 \tag{3}$$

can be written in the formulation

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\frac{1}{\rho}\nabla \cdot \boldsymbol{\sigma}$$
$$\boldsymbol{\sigma} = -pI + \mu(\nabla \mathbf{v} + \nabla \mathbf{v}^T) \tag{4}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor. Taking trace both sides in the second equation of 4 and using the divergence free condition for velocity, we get $p = -tr\boldsymbol{\sigma}/2$.

Streamlines and Stream functions

Streamlines are a family of curves whose tangent vectors constitute the velocity vector field of the flow. For a streamline, $\mathbf{ds} \times \mathbf{v} = 0$ where **ds** is a differential element. Equation of a streamline in two dimensions of velocity **v** with components (u, v)

$$\frac{dy}{dx} = \frac{v}{u} \implies v(dx) - u(dy) = 0$$

The stream function ψ is the volume flux through a streamline between points A and P

$$\psi = \int_A^P (udy - vdx)$$

An infinitesimal shift in $\delta P = (\delta x, \delta y)$ of the position P results in a change of the stream function $\delta\psi = u\delta y - v\delta x$ and from the exact differential $\delta\psi = \frac{\partial\psi}{\partial x}\delta x + \frac{\partial\psi}{\partial y}\delta y$ the flow velocity components:

$$u = \frac{\partial\psi}{\partial y} \quad v = -\frac{\partial\psi}{\partial x} \tag{5}$$

$$\nabla \cdot \boldsymbol{\sigma} = \begin{pmatrix} \frac{\partial}{\partial x}(-p + 2\mu \frac{\partial u}{\partial x}) & \frac{\partial}{\partial y}(\mu \frac{\partial u}{\partial y} + \mu \frac{\partial v}{\partial x}) \\ \frac{\partial}{\partial x}(\mu \frac{\partial u}{\partial y} + \mu \frac{\partial v}{\partial x}) & \frac{\partial}{\partial y}(-p + 2\mu \frac{\partial v}{\partial y}) \end{pmatrix} \tag{6}$$

Hence the momentum equation is given by

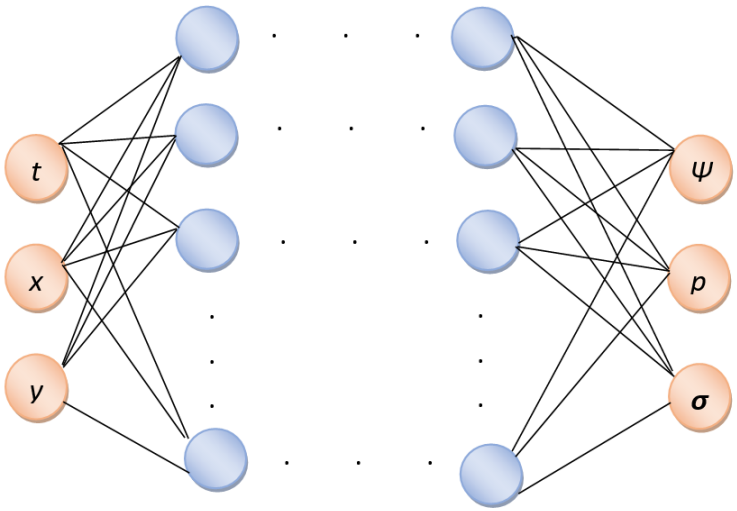
$$\begin{pmatrix} f_u \\ f_v \end{pmatrix} = \begin{pmatrix} \rho \frac{\partial u}{\partial t} + \rho(u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y}) - \frac{\partial}{\partial x}(-p + 2\mu \frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(\mu \frac{\partial u}{\partial y} + \mu \frac{\partial v}{\partial x}) \\ \rho \frac{\partial v}{\partial t} + \rho(u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y}) - \frac{\partial}{\partial y}(-p + 2\mu \frac{\partial v}{\partial y}) - \frac{\partial}{\partial x}(\mu \frac{\partial u}{\partial y} + \mu \frac{\partial v}{\partial x}) \end{pmatrix} \tag{7}$$

and the pressure p is given as

$$p = -tr(\boldsymbol{\sigma})/2 = -\frac{1}{2}((-p + 2\mu \frac{\partial u}{\partial x}) + (-p + 2\mu \frac{\partial v}{\partial y})) \tag{8}$$

Navier Stokes using PINN

- We train a Deep Neural Network(DNN) that maps the spatio-temporal variable $\{t, \mathbf{x}\}^T$ to the mix-variable solution $\{\psi, p_{pred}, \boldsymbol{\sigma}\}$, p_{pred} is the predicted pressure.



- $u_{pred} = \frac{\partial \psi}{\partial y} \quad v_{pred} = -\frac{\partial \psi}{\partial x}$

- The output $\boldsymbol{\sigma}$ has three components $\boldsymbol{\sigma} = \{s_{11}, s_{22}, s_{12}\}^T$.

$$\begin{pmatrix} f_{upred} \\ f_{vpred} \end{pmatrix} := \begin{pmatrix} \rho \frac{\partial u_{pred}}{\partial t} + \rho(u_{pred} \frac{\partial u_{pred}}{\partial x} + v_{pred} \frac{\partial u_{pred}}{\partial y}) - \frac{\partial s_{11}}{\partial x} - \frac{\partial s_{12}}{\partial y} \\ \rho \frac{\partial v_{pred}}{\partial t} + \rho(u_{pred} \frac{\partial v_{pred}}{\partial x} + v_{pred} \frac{\partial v_{pred}}{\partial y}) - \frac{\partial s_{22}}{\partial y} - \frac{\partial s_{12}}{\partial x} \end{pmatrix} \tag{9}$$

- On comparing 7 and 9 we see that u_{pred}, v_{pred} at $\{x, y, t\}$ can satisfy the momentum equation if s_{11}, s_{22} and s_{12}, each of them approximate $(-p_{pred} + 2\mu \frac{\partial u_{pred}}{\partial x})$, $(-p_{pred} + 2\mu \frac{\partial v_{pred}}{\partial y})$ and $(\mu \frac{\partial u_{pred}}{\partial y} + \mu \frac{\partial v_{pred}}{\partial x})$ respectively.

$$\begin{pmatrix} f_{s_{11}} \\ f_{s_{12}} \\ f_{s_{22}} \end{pmatrix} := \begin{pmatrix} (-p_{pred} + 2\mu \frac{\partial u_{pred}}{\partial x}) - s_{11} \\ (\mu \frac{\partial u_{pred}}{\partial y} + \mu \frac{\partial v_{pred}}{\partial x}) - s_{12} \\ (-p_{pred} + 2\mu \frac{\partial v_{pred}}{\partial y}) - s_{22} \end{pmatrix} \tag{10}$$

- p_{pred} must satisfy 8, that is, $p_{pred} = -\frac{1}{2}(s_{11} + s_{22})$.

$$f_{ppred} := p_{pred} + \frac{1}{2}(s_{11} + s_{22}) \tag{11}$$

Loss Function

Let r_g denote the residual at $\{x, y, t\}$ obtained from the governing equations Then

$$||r_g(x, y, t)||^2 = |f_{upred}|^2 + |f_{vpred}|^2 + |f_{ppred}|^2 + |f_{s_{11}}|^2 + |f_{s_{12}}|^2 + |f_{s_{22}}|^2$$

Let r_i, r_b denote the residuals arising from the initial and boundary conditions and contribute to the physical loss at all the domain points. If $\{x, y, 0\}$ are the domain points at time $t = 0$ then

$$\begin{aligned} ||r_I(x, y, 0)||^2 := &||u(x, y, 0) - u_{pred}(x, y, 0)||^2 + ||v(x, y, 0) - v_{pred}(x, y, 0)||^2 \\ &+ ||p(x, y, 0) - p_{pred}(x, y, 0)||^2 \end{aligned}$$

and for any point (x, y) belonging to the boundary at time t

$$\begin{aligned} ||r_b(x, y, t)||^2 := &||u(x, y, t) - u_{pred}(x, y, t)||^2 + ||v(x, y, t) - v_{pred}(x, y, t)||^2 \\ &+ ||p(x, y, t) - p_{pred}(x, y, t)||^2 \end{aligned}$$

Loss: The loss function is composed of the physical loss J_g obtained from the governing equations and the initial boundary condition loss $J_{i/bc}$ given by

$$\begin{aligned} J_g &= \frac{1}{N_g}\sum_{i=1}^{N_g} ||r_g(x^i, y^i, t^i)||^2 \\ J_{i/bc} &= \frac{1}{N_I}\sum_{i=1}^{N_I} ||r_I(x^i, y^i, 0)||^2 + \frac{1}{N_b}\sum_{i=1}^{N_b} ||r_b(x^i, y^i, t^i)||^2 \end{aligned} \tag{12}$$

where $N_{(.)}$ denotes the number of collocation points.

The total physical loss J_p is defined as

$$J_p = J_g + \beta J_{i/bc} \tag{13}$$

where $\beta > 0$ is a user defined weighting coefficient for the initial and boundary condition loss.

At every epoch, the weights are updated by using various optimization algorithms like Adam, BFGS or L-BFGS.

Test Examples

Example 1:

- The computational domain is a rectangle in two dimension of length 1.1 cm and breadth 0.41 cm.
- The viscosity $\nu = 0.01 gs^{-1} cm^{-1}$.
- The time duration for simulation/modelling is $T = 0.5$ seconds with the time step $\Delta t = 0.01$ seconds.
- **Boundary Conditions**

$$\mathbf{u} = 0 \qquad \text{on } \Gamma_D$$

$$\mathbf{u} = \begin{pmatrix} 2 \cdot x_1 \frac{0.41 - x1}{0.41^2} \cdot (sin(\frac{\pi t}{T} + \frac{3\pi}{2}) + 1) \\ 0 \end{pmatrix} \qquad \text{on } \Gamma_{in}$$

$$p = 0 \qquad \text{on } \Gamma_{out}$$

- **Initial conditions** $(\mathbf{u}^0, p^0) = (\mathbf{0}, 0)$

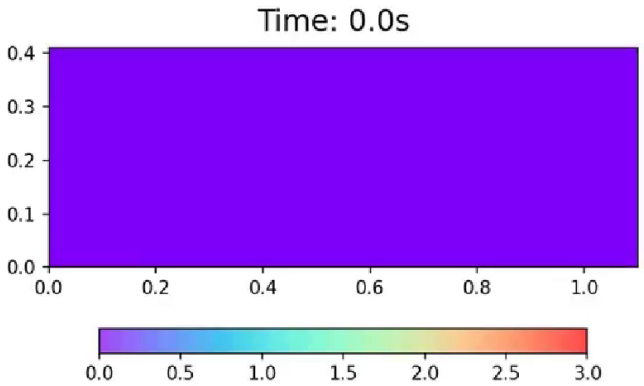
Simulation using PINNs

- PINNs is implemented in TensorFlow[6]. The network has 3 neurons in the input layer, 5 neurons in the output layer and 7 hidden layers with 50 neurons each.
- The activation function used for each hidden layer is *tanh* activation function. The coefficient β is set to 2.
- A total number of 3321 collocation points which include $N_b = 244$ points on the boundary, $N_{in/out} = 81$ on the inlet/outlet boundary is used for training.
- For 5000 iterations, Adam optimizer is used and L-BFGS optimizer is used for 85480 iterations. The maximum number of iterations to find optimal step length α_k using Hager-Zhang line search algorithm is 50.
- The total amount of time taken to train the network is 159045.8855 seconds.
- A total number of 64561 collocation points which include $N_b = 1124$ points in the boundary $N_{in/out} = 161$ on the inlet/outlet boundary is used for prediction. The training and prediction is carried out by $\beta = 5$, $\beta = 10$ and the loss is plotted using python.

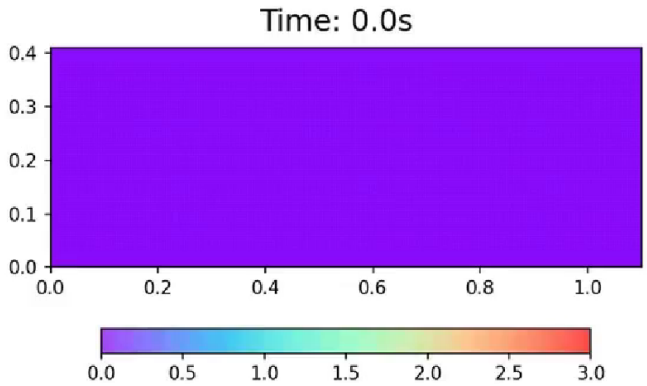
Simulation using DUNE

- Simulation using Finite Element method is carried out in DUNE[1] Framework. The geometry is created using GMSH[4] and an Unstructured grid is used for meshing.
- The velocity is of degree two and pressure is taken to be of degree one.
- The number of elements in the mesh is 32724. The number of edges in the mesh is 97445. Total number of vertices in the mesh is 64722. The number of constrained degrees of freedom is 2254.
- Crank Nicholson Scheme is used for discretization. The non linear solver used is Newton. Bi-Conjugate Gradient Stabilized with ILU Preconditioner is used as the linear solver.
- The CPU time taken for the simulation is 3374.51 seconds.

Velocity Simulation

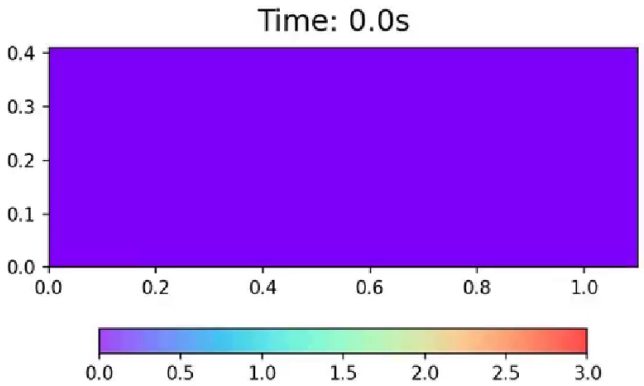


DUNE

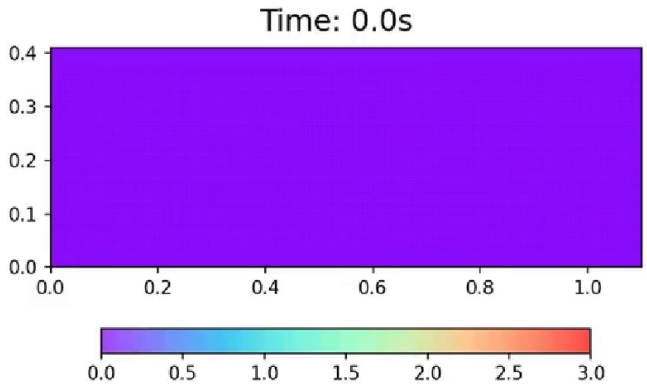


PINN

Pressure Simulation

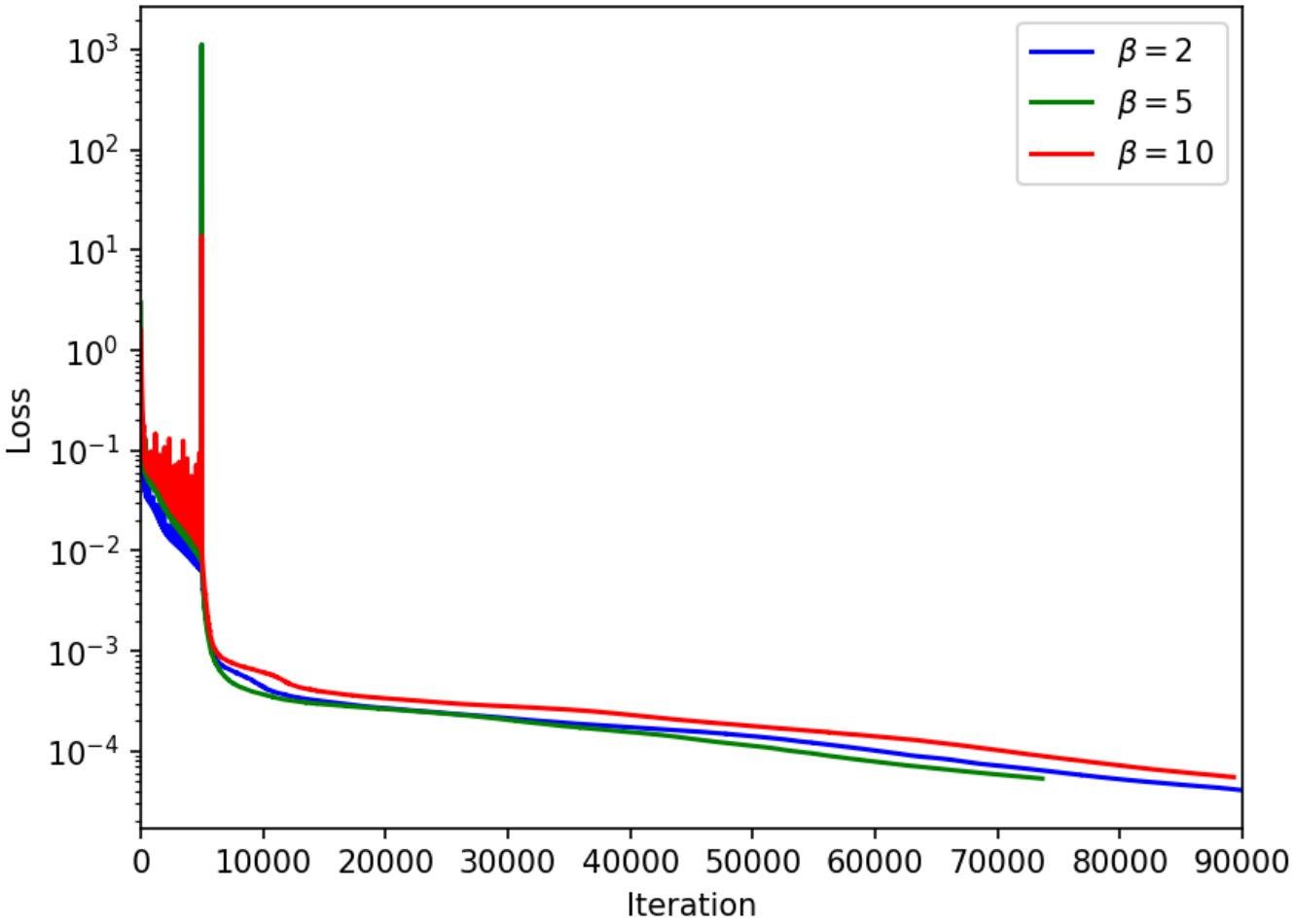


DUNE



PINN

Comparison of loss curves for different β



Observations

- The results obtained through PINNs and DUNE agree qualitatively.
- The loss produced is 4.02482×10^{-5} for $\beta = 2$. The rate of convergence of L-BFGS optimizer is faster for $\beta = 5$.

Example 2:

- The computational domain is a semicircular pipe with smooth disturbances in two dimension with cross sectional radius $a = 1.6cm$ and curvature radius $R = 2.9cm$.
- The viscosity $\nu = 0.4gs^{-1}cm^{-1}$.
- The time duration for simulation/modelling is $T = 6$ seconds with the time step $\Delta t = 0.01$ seconds.
- **Boundary Conditions**

$$\mathbf{u} = 0 \qquad \text{on } \Gamma_D$$

$$\mathbf{u} = \begin{pmatrix} 0 \\ 3 \cdot x_1 \cdot \frac{3.2 - x_1}{3.2^2} \cdot (sin(\frac{\pi t}{T} + \frac{3\pi}{2}) + 1) \end{pmatrix} \qquad \text{on } \Gamma_{in}$$

$$p = 0 \qquad \text{on } \Gamma_{out}$$

- **Initial conditions** $(\mathbf{u}^0, p^0) = (\mathbf{0}, 0)$

Simulation using PINNs

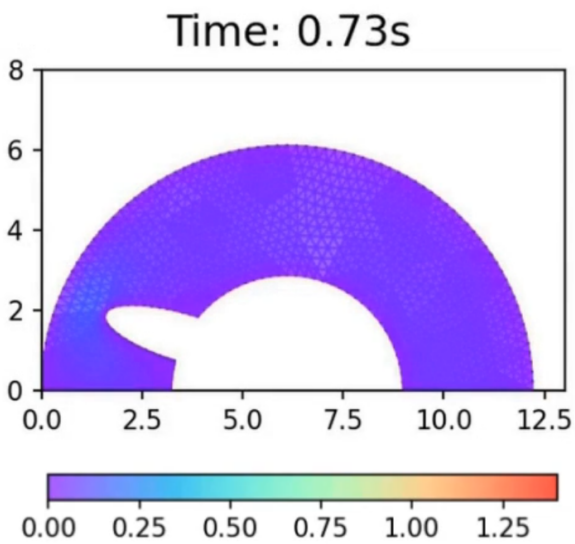
- PINNs is implemented in TensorFlow[6]. The network has 3 neurons in the input layer, 5 neurons in the output layer and 7 hidden layers with 50 neurons each.
- The activation function used for each hidden layer is *tanh* activation function. The coefficient β is set to 2.
- The total number of collocation points is 29760 and a batch of 20000 points is used in every iteration to train the network.
- For 800 iterations, Adam optimizer is used and L-BFGS optimizer is used for 11344 iterations. The maximum number of iterations to find optimal step length α_k using Hager-Zhang line search algorithm is 50.
- The total amount of time taken to train the network is 94102.45567 seconds.
- A total number of 29760 collocation points is used for prediction and the loss is plotted using python.

Simulation using DUNE

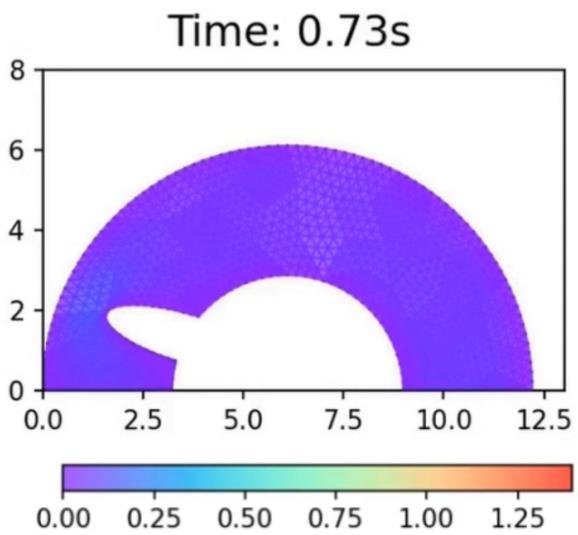
- Simulation using Finite Element method is carried out in DUNE[1] Framework. The geometry is created using GMSH[4] and an Unstructured grid is used for meshing.
- The velocity is of degree two and pressure is taken to be of degree one.
- The number of elements in the mesh is 1984. The number of edges in the mesh is 3064. Total number of vertices in the mesh is 1081. The number of constrained degrees of freedom is 659.
- Crank Nicholson Scheme is used for discretization. The non linear solver used is Newton. Bi-Conjugate Gradient Stabilized with ILU Preconditioner is used as the linear solver.
- The CPU time taken for the simulation is 131.479 seconds.

Velocity Simulation

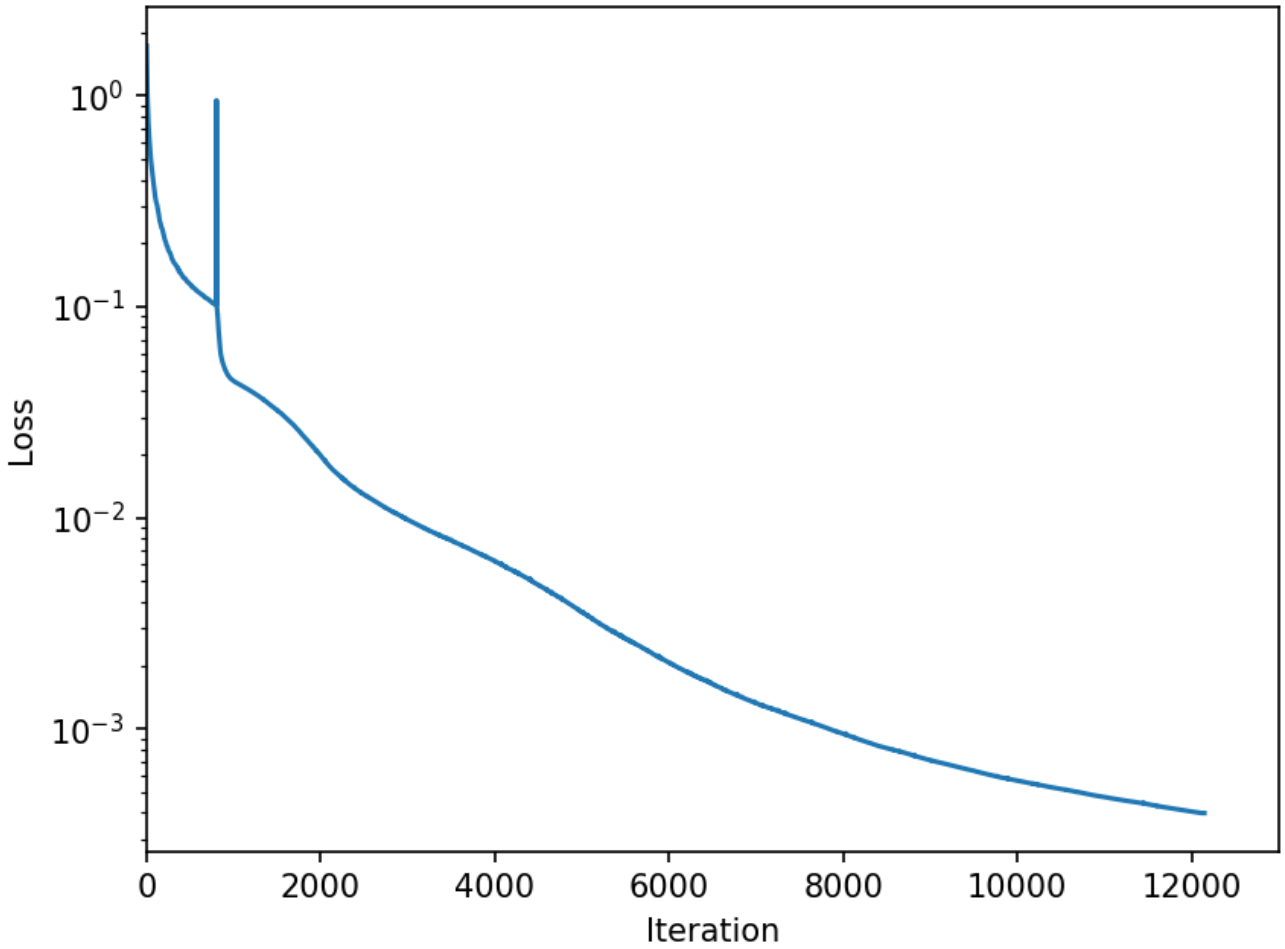
DUNE



PINN



Loss curve for Example 2 with $\beta = 2$



Observations

- The results obtained through DUNE and PINNs agree qualitatively.
- The loss produced is 3.991×10^{-4} for $\beta = 2$.

Table of Contents

Conclusion

- We have modeled the blood flow in an open, bounded Lipschitz domain through the Navier-Stokes Equations and studied the existence and uniqueness of a solution in two dimensions through FEM.
- Derived a finite element formulation of the Navier-Stokes Equation in appropriate finite-dimensional velocity and pressure spaces, which are the discrete counterparts of continuous spaces $\mathbf{H}_0^1(\Omega)$ and $L^2(\Omega)$.
- Examined stable and unstable pairs of Finite Element Spaces and proved that the finite element pairs P_1/P_1 pair and P_1/P_0 are unstable. The Taylor Hood finite element pair P_{r+1}/P_r for $r \geq 1$ was found to be stable and was used in the finite-dimensional formulation of the Navier-Stokes Equation.
- Discussed and simulated the blood flow in circular domains using Finite Element Method in two and three dimensions. The effect of Dean number on flows in a circular domain was demonstrated.

- Implemented the mixed-variable PINNs formulation proposed in [2].
- The solution got through PINNs and using FEM closely agreed qualitatively on straight domains. The CPU time for training PINNs on a straight domain is 159045.8855 seconds with 3321 collocation points which are computationally expensive compared to the solution obtained in 3374.51 seconds using FEM. The loss is minimized using ADAM Optimization Algorithm for 5000 iterations followed by the LBFGS Optimization Algorithm for 85840 iterations to achieve a loss of 4.024×10^{-5}.
- The results on the circular domain obtained through PINNs agreed qualitatively with FEM. The training time for PINNs is 94102.4556 seconds for a batch size of 20000 at every epoch compared to the solution in 131.479 seconds using FEM. The loss is minimized using the ADAM optimization algorithm for 800 iterations followed by the LBFGS optimization algorithm for 11344 iterations to achieve a loss of 3.991×10^{-4}.

Future Work

- Parallel programming with GPU can significantly reduce the training time to provide a promising mesh-free technique for predicting blood flow in various domains and has been left for future work.

Thank You

Acknowledgements

- Dr Nagaiah Chamakuri
- Scientific Computing Research group
- Friends and Family

References I

Dune numerics.

https://www.dune-project.org/.

Chengping Rao, Hao Sun, and Yang Liu.

Physics-informed deep learning for incompressible laminar flows.

Theoretical and Applied Mechanics Letters, 10:207–212, 2020.

Auricchio Lefieux.

Computational study of aortic hemodynamics: From simplified to patient-specific geometries.

Advances in Computational Fluid-Structure Interaction and Flow Simulation (pp.397-407), 2016.

References II

A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities.

https://gmsh.info/.

Kratzke and Mang.

Tutorial: Aortic blood flow simulation using hiflow.

https://emcl-gitlab.iwr.uni-heidelberg.de/hiflow3.org/hiflow3/-/wikis/uploads/ff8030ea189d71d6b546426a50e6a52a/tut_blood_flow.pdf.

Tensorflow.

https://www.tensorflow.org/.

References III

V. John.

Finite Element Methods for Incompressible Flow Problems.

Springer Series in Computational Mathematics. Springer International Publishing, 2016.

H.C. Elman, D.J. Silvester, and A.J. Wathen.

Finite Elements and Fast Iterative Solvers:with Applications in Incompressible Fluid Dynamics: with Applications in Incompressible Fluid Dynamics.

Numerical Mathematics and Scientific Computation. OUP Oxford, 2005.

V Girault and P A Raviart.

Finite Element Methods for Incompressible Flow Problems.

Second Edition. Springer, 1989.

References IV

R. Temam.

Navier-Stokes Equations: Theory and Numerical Analysis.

Studies in mathematics and its applications. North-Holland, 1984.

L. Formaggia, A. Quarteroni, and A. Veneziani.

Cardiovascular Mathematics: Modeling and simulation of the circulatory system.

MS&A. Springer Milan, 2010.

J. Nocedal and S. Wright.

Numerical Optimization.

Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

References V

H. Jiang.

Machine Learning Fundamentals.

Cambridge University Press, 2021.

Justine Fouchet-Incaux.

Artificial boundaries and formulations for the incompressible navier–stokes equations: applications to air and blood flows.

SeMA, 2014.

Yogesh Karnam.

Multiscale fluid-structure interaction models development and applications to the 3d elements of a human cardiovascular system.

RIT Scholar Works, Rochester Institute of Technology, 2019.

References VI

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.

Laminar flow.

https://www.simscale.com/docs/simwiki/cfd-computational-fluid-dynamics/what-is-laminar-flow/.

Physics informed neural networks.

https://github.com/maziarraissi/PINNs.

Pinn-laminar-flow.

https://github.com/Raocp/PINN-laminar-flow.

References VII

Automatic differentiation in machine learning: a survey.

https://arxiv.org/pdf/1502.05767.pdf.

Automatic differentiation background.

https://in.mathworks.com/help/deeplearning/ug/deep-learning-with-automatic-differentiation-in-matlab.html.

Bfgs algorithm.

https://en.wikipedia.org/wiki/Broyden-Fletcher-Goldfarb-Shanno_algorithm, https://en.wikipedia.org/wiki/Quasi-Newton_method.

Machine learning lecture notes – mit 6.390 fall 2022.

https://introml.mit.edu/_static/fall22/LectureNotes/6_390_lecture_notes_fall2022.pdf.

Table of Contents

Mini Batch Gradient Descent

Mini-batch SGD replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of data).

Algorithm 6 mini-batch SGD Algorithm

randomly initialize weights W for every layer and initial learning rate α_0
set epoch $t = 0$ and update $n = 0$
while not converged **do**
  randomly shuffle train data into mini batches
  for each mini batch B and each layer l **do**
    for each $\mathbf{x} \in B$ **do**
      forward pass $\mathbf{x} \longrightarrow \mathbf{y}$
      backward pass $\{\mathbf{x}, \mathbf{y}\} \longrightarrow \frac{\partial C(W_n^{[l]};\mathbf{x})}{\partial W_n^{[l]}}$
    end for
    update model: $W_{n+1}^{[l]} = W_n^{[l]} - \frac{\alpha_t}{|B|} \sum_{x \in B} \frac{\partial C(W_n^{[l]};\mathbf{x})}{\partial W^{[l]}}$
    n=n+1
  end for
  $\alpha_t \longrightarrow \alpha_{t+1}$
  $t = t + 1$

Adam's Optimization Algorithm

Running Averages: If the sequence of data is $\alpha_1, \alpha_2, \ldots$ then the sequence of running averages $A_0, A_1, A_2, \ldots$ can be defined as $A_0 = 0$ and $A_t = \gamma_t A_{t-1} + (1 - \gamma_t)\alpha_t$ where $\gamma_t \in (0, 1)$. If γ_t is a constant, then this is a moving average in which,

$$A_T = \sum_{i=1}^{T} \gamma^{T-t}(1 - \gamma)\alpha_t$$

The inputs α_t closer to the end of the sequence have more effect on A_T than early inputs.

Adaptive Moment Estimation: ADAM Algorithm is a method for stochastic optimization that only requires first order gradients and uses the estimations of the first and second moments of the gradient to adapt the learning rate for each weight of the neural network.

Algorithm 7 ADAM Algorithm

randomly initialize weights W for every layer and initial learning rate α_0

set epoch $t = 0$, $n = 0$ and $\mathbf{u}_0 = \mathbf{v}_0 = 0$

while not converged **do**

randomly shuffle train data into mini batches

for each mini batch B and each layer l **do**

for each $\mathbf{x} \in B$ **do**

forward pass $\mathbf{x} \longrightarrow \mathbf{y}$

backward pass $\{\mathbf{x}, \mathbf{y}\} \longrightarrow \frac{\partial C(W_n^{[l]};\mathbf{x})}{\partial W^{[l]}}$

end for

$\mathbf{g_n} = \frac{\alpha_t}{|B|} \sum_{x \in B} \frac{\partial C(W_n^{[l]};\mathbf{x})}{\partial W^{[l]}}$

$\mathbf{u}_{n+1} = \alpha \mathbf{u}_n + (1 - \alpha)\mathbf{g_n}$

$\mathbf{v}_{n+1} = \beta \mathbf{v}_n + (1 - \beta)\mathbf{g_n} \odot \mathbf{g_n}$

$\hat{\mathbf{u}}_{n+1} = \frac{\mathbf{u}_{n+1}}{1-\alpha^{n+1}}$ and $\hat{\mathbf{v}}_{n+1} = \frac{\mathbf{v}_{n+1}}{1-\beta^{n+1}}$

update model: $W_{n+1}^{[l]} = W_n^{[l]} - \eta \hat{\mathbf{u}}_{n+1} \odot ((\hat{\mathbf{v}}_{n+1} + \epsilon^2)^{-\frac{1}{2}})$

n=n+1

end for

$\alpha_t \longrightarrow \alpha_{t+1}$

t=t+1

BFGS Algorithm

For an unconstrained smooth convex optimization $\min_x f(x)$ where f is convex, twice differentiable, the Newton's Method update is given as

$$x_{k+1} = x_k + (B(x_k))^{-1}(\nabla f(x_{k+1}) - \nabla f(x_k))$$

Newton method has quadratic convergence but computing the Hessian matrix B at x_k and solving the system $B(x_k)\Delta x = (\nabla f(x_{k+1}) - \nabla f(x_k))$ at every iteration is computationally expensive.

BFGS method computes the inverse of B_k, denoted by $H_k = B_k^{-1}$ by solving the following optimization problem

$$\min_H ||H - H_k||$$

$$\text{subject to } H = H^T, \quad H(x_k)(\nabla f(x_{k+1}) - \nabla f(x_k)) = \Delta x$$

Algorithm 8 BFGS Algorithm

1: Given starting point x_0, tolerance ϵ, inverse Hessian approximation H_0
2: set $k = 0$
3: **while** $||\nabla f_k|| > \epsilon$ **do**
4: Compute the search direction $p_k = -H_k \nabla f_k$
5: Compute α_k from a line search procedure to satisfy the Wolfe conditions.
6: Set $x_{k+1} = x_k + \alpha_k p_k$
7: Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$
8: Compute $H_{k+1} = (I - \frac{s_k y_k^T}{y_k^T s_k}) H_k (I - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k}$
9: $k = k + 1$

Algorithm 4 BiCGSTAB with ILU Preconditioning

Input A, $\mathbf{b}$, tolerance tol, max_iter, fill-in parameter p

Compute ILU factorization of A: L, $U = \text{ILU}(A, p)$.

Initialize $k = 0$, $\mathbf{x}_0 = 0$, $\mathbf{r}_0 = b$, $\tilde{\mathbf{r}}_0 = b$, $\mathbf{p}_0 = 0$, $\tilde{\mathbf{p}}_0 = 0$

$\tilde{\mathbf{z}}_0 := U^{-1}\tilde{\mathbf{r}}_0$

$\mathbf{z}_0 := L^{-1}\mathbf{r}_0$

$\rho_0 := \tilde{\mathbf{z}}_0^T \mathbf{r}_0$

while True **do**

$k = k + 1$

$\alpha_k = \frac{\rho_{k-1}}{(\tilde{\mathbf{p}}_{k-1}^T A \mathbf{p}_{k-1})}$.

$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_{k-1}$

$\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k A \mathbf{p}_{k-1}$

if $\mathbf{r}_k <$tol **then**

break

end if

$\tilde{\mathbf{r}}_k = \tilde{\mathbf{r}}_{k-1} - \alpha_k A^T \tilde{\mathbf{p}}_{k-1}$

$\mathbf{z}_k = L^{-1}\mathbf{r}_k$

$\tilde{\mathbf{z}}_k = U^{-1}\tilde{\mathbf{r}}_k$

$\rho_k = \tilde{\mathbf{z}}_k^T \mathbf{r}_k$

$\beta_k = \frac{\rho_k}{\rho_{k-1}}$.

$\mathbf{p}_k = \mathbf{z}_k + \beta_k \mathbf{p}_{k-1}$

$\tilde{\mathbf{p}}_k = \tilde{\mathbf{z}}_k + \beta_k \tilde{\mathbf{p}}_{k-1}$

end while

Algorithm 3 ILU factorization with fill-in parameter p

Input A, fill-in parameter p

n is the size of matrix A

Set $L = U = [0]_{n \times n}$ ▷ Initialize matrices to zero

for $k = 1$ to n **do**

$U[k,k] = A[k,k] - \sum_{j=1}^{k-1} L[k,j] \cdot U[j,k]$ ▷ Diagonal entry of U

Compute non-zero entries above the diagonal of U:

for $j = k+1$ to $\min(k+p, n)$ **do**

$U[k,j] = \frac{1}{U[k,k]} \left(A[k,j] - \sum_{i=1}^{k-1} L[k,i] \cdot U[i,j] \right)$

end for

$L[k,k] = 1$ ▷ Diagonal entry of L

Compute non-zero entries below the diagonal of L:

for $i = k+1$ to $\min(k+p, n)$ **do**

$L[i,k] = \frac{1}{U[k,k]} \left(A[i,k] - \sum_{j=1}^{k-1} L[i,j] \cdot U[j,k] \right)$

end for

end for

Return L and U