Data assimilation in cardiovascular modelling using physics-informed neural networks

Alessandro Danesi

February 10, 2021

1 Introduction

Nowadays it is more and more important understanding the inner workings of the cardiovascular system in order to have a valuable tool for monitoring, diagnostics and surgical planning that can be deployed on a large patient-specific topologies of system arterial networks.
Although the collected in-vivo measurements can be highly accurate, such interventional techniques are sometimes expensive and suffer from limitations that are not easy to address, e.g., difficulties of placing probes incerebral or uteroplacental arteries.
In this work I put forth a machine learning framework that enables the seamless synthesis of non-invasive in-vivo measurement techniques and computational flow dynamics models derived from first physical principles.
Specifically, I propose to employ deep neural networks to represent the unknown flow variables (blood velocity, wall displacement and pressure) in a given arterial network.
I then train these networks to produce outputs that:

- fit any available clinical data
- satisfy the underlying physical conservation described by a reduced 1D model of pulsatile blood flow
- ensure conservation and information propagation across interface points in the network

In contrast to traditional computational fluid dynamics models, the proposed methodology does not involve the cumbersome generation of computational meshes, nor it requires the precise prescription of boundary conditions.
Instead, it can directly use noisy and scattered measurements obtained by medical imaging or by CFD solvers at points other than the boundaries to accurately predict the pressure and recover flow information. Furthermore, as flow conditions in general do not vary greatly from one patient to another, pre-trained neural network models can be rapidly adapted to a new patient case. After the model is trained, all quantities of interest (blood velocity, wall displacement and pressure) can be predicted for every temporal or spatial point in the network.

2 Methods

2.1 A simplified model of pulsatile flow in arterial network

In order to model the pulse wave propagation in arterial networks I deployed 1D reduced models with some assumptions, such as considering the local curvature small, the fluid incompressible and Newtonian since we are considering large arteries and the blood density and dynamic viscosity constant.

The following equations have been used:

$$\begin{aligned}
&\frac{\partial A}{\partial t} + \frac{\partial Au}{\partial x} = 0 \\
&\frac{\partial u}{\partial t} + \alpha u \frac{\partial u}{\partial x} + \frac{u}{A}\frac{\partial (\alpha - 1)uA}{\partial x} + \frac{1}{\rho}\frac{\partial p}{\partial x} - K_R \frac{u}{A} = 0 \\
&p = p_{ext} + \beta(\sqrt{A} - \sqrt{A_0})
\end{aligned} \tag{1}$$

The first one is an hyperbolic conservation law that describes the evolution of blood velocity and cross-sectional area.
The second one, instead, is a reduced form of the incompressible Navier-Stokes equations. In particular, K_R is a friction parameter that depends on the velocity profile chosen, ρ the density of blood while α is a momentum flux correction factor which accounts for the non-linearity of the sectional integration in terms of the local velocity (in the following numerical examples, the viscous losses coefficient K_R was set to zero and α to one under the assumption that viscous losses play a secondary role in the larger systemic arteries considered in this work.).
Finally, last equation shows the relation between the pressure and the area using Laplace's law. Here, p_{ext} is the diastolic pressure and $\beta = \frac{\sqrt{\pi} h_0 E}{(1-\nu^2)A_0}$, where h_0 is the wall thickness, E the Young's modulus, ν the Poisson ratio and A_0 the wall displacement.
This reduced order model provides an accurate representation of the underlying transport processes and its effectiveness in correctly capturing pulse wave propagation phenomena has been validated against both in-vitro and in-vivo data.

2.2 Physics-informed neural networks

I introduce now the physics-informed neural networks, namely neural networks that are trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations.
In this particular case, I define one neural network $f(x, t; \theta^j)$ to represent the solution of Eq.(1) for vessel j in the arterial network. Here θ^j denotes the parameters of the neural network for vessel j. The network aims to approximate the following mapping:

$$[x, t] \mapsto [A^j(x, t), u^j(x, t), p^j(x, t)]$$

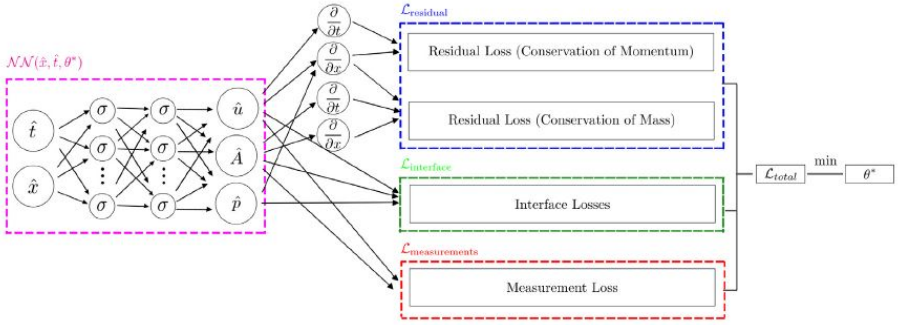

Figure 1: A schematic representation of the proposed algorithm.

In correspondence to Eq.(1), the following residuals can be defined:

$$
\begin{aligned}
r_A(x,t) &:= \frac{\partial A}{\partial t} + \frac{\partial Au}{\partial x} \\
r_u(x,t) &:= \frac{\partial u}{\partial t} + \alpha u \frac{\partial u}{\partial x} + \frac{u}{A}\frac{\partial(\alpha-1)uA}{\partial x} + \frac{1}{\rho}\frac{\partial p}{\partial x} - K_R\frac{u}{A} \\
r_p(x,t) &= p - p_{ext} - \beta(\sqrt{A} - \sqrt{A_0})
\end{aligned}
\tag{2}
$$

These residuals can be then used as constraints during the training of the neural networks $f(x,t;\theta^j)$ in order to encourage them to produce physically consistent predictions.
In addition to minimizing the residuals, the neural networks are also constrained to fit any available non-invasive clinical measurements that may be noisy and scattered in space and time. Going into deeper details, for the vessel j, for instance, I considered $\{(x_i,t_i), A^j(x_i,t_i)\}, i = 1,...,N_A^j$ and $\{(x_i,t_i), u^j(x_i,t_i)\}, i = 1,...,N_u^j$ respectively for cross sectional area and blood velocity, along with a larger number of collocation points $\{(x_i,t_i), r^j_{A,u,p}(x_i,t_i) = 0\}, i = 1,...,N_r^j$ that try to satisfy residual equations (2).

2.3 Loss functions

2.3.1 Measurements

This part of the loss function corresponds to fitting the clinical measurements obtained for some of the vessels in the network. In this particular case, this term encourages the output of the neural network A and u to match the measurements of area and velocity. This part of the loss function has the form (take vessel j as example):

$$
\mathcal{L}^j_{measurements} = \frac{1}{N_u^j}\sum_{i=1}^{N_u^j}(u^j(x_i,t_i) - u^j(x_i,t_i;\theta^j))^2 + \frac{1}{N_A^j}\sum_{i=1}^{N_A^j}(A^j(x_i,t_i) - A^j(x_i,t_i;\theta^j))^2
$$

for $j = 1, ..., D_M$

N_A^f, N_u^f represent the number of measurements for A and u in vessel j respectively. Also, D_M denotes total number of vessels in which we have measurements. In the above equation $u^j(x_i, t_i; \theta^j)$ and $A^j(x_i, t_i; \theta^j)$ represent the outputs given the parameters of the neural network for vessel j. Minimizing this term will encourage the neural networks to fit the available measurements.

2.3.2 Collocation points

This part of the loss function corresponds to the collocation points, chosen randomly inside the arterial domains using a latin-hypercube sampling strategy. Over these collocation points, we impose the physical constraints by encouraging the right hand side of Eq.(2) to be equal to zero. This objective is imposed to encourage the neural networks to find a particular set of parameters that make their predictions consistent with the underlying differential equations, which translates to having the minimum residual. This collocation loss function takes the following form (take vessel j as example):

$$\mathcal{L}^j_{residual} = \frac{1}{N_r^j}\sum_{i=1}^{N_r^j}(r_A^j(x_i, t_i; \theta^j)^2 + \frac{1}{N_r^j}\sum_{i=1}^{N_r^j}(r_u^j(x_i, t_i; \theta^j)^2 + \frac{1}{N_r^j}\sum_{i=1}^{N_r^j}(r_p^j(x_i, t_i; \theta^j)^2$$

for $j = 1, ..., D$

N_r^f represents the number of collocation points in vessel j. Also, D denotes the total number of vessels in the arterial network.
The terms $r_A^j(x_i, t_i; \theta^j), r_u^j(x_i, t_i; \theta^j), r_p^j(x_i, t_i; \theta^j)$ represent the residual of area, velocity and pressure defined in Eq.(2), respectively.

2.3.3 Interfaces

Thanks to neural networks, it is possible to impose the momentum and mass conservation equations without using information on the characteristics in order to ensure conservation on boundaries. In particular, if I consider a bifurcation where a vessel splits into two other vessels, I can use conservation of momentum and mass for the parent and daughter vessels as the following:

$$\begin{aligned} A_1 u_1 &= A_2 u_2 + A_3 u_3 \\ p_1 + \frac{1}{2}\rho u_1^2 &= p_2 + \frac{1}{2}\rho u_2^2 \\ p_1 + \frac{1}{2}\rho u_1^2 &= p_3 + \frac{1}{2}\rho u_3^2 \end{aligned} \tag{3}$$

In the case of more than one bifurcations, these conditions are imposed to every splitting point using the appropriate notation, depending on the numbering of the domains.

2.3.4 Final loss function

Finally, I can combine these individual terms to create the joint form of the loss function:

$$\mathcal{L} = \sum_{j=1}^{D_M} \mathcal{L}^j_{\text{measurements}} + \sum_{j=1}^{D} \mathcal{L}^j_{\text{residual}} + \sum_{k=1}^{D_I} \mathcal{L}^k_{\text{interfaces}}, \tag{4}$$

This is the complete form of the loss function that encourages the model to learn the underlining physics of the problem by using measurements inside the domain, not necessarily at the boundaries, and by respecting the imposed physical constraints.

2.4 Non-dimensionalization and normalization

In Eq.(1), the order of magnitude of the different physical quantities, velocity, cross-sectional area and pressure, have a significant relative difference, e.g., $P \sim 10^6\ Pa$, $A \sim 10^{-5}\ m^2$ and $u \sim 10\ m/s$, which casts great difficulty on the training of the neural network. The significant difference in magnitude of the parameters creates a systematic problem for the training of the physics-informed neural network, as this difference in scales will have a severe impact on the magnitude of the back-propagated gradients that adjust the neural network parameters during training. To overcome this problem, it can be employed a non-dimensionalization and normalization technique with the purpose of scaling the input and the output of the neural networks in a proper scale and normalizing the spatial and temporal coordinates to have zero mean and unit variance for training the neural networks more efficiently.
For this purpose, as suggested I have chosen the characteristic length to be the square root of the mean of the equilibrium cross-sectional area of the network vessels and the characteristic velocity equal to 10, one order of magnitude larger than the length. Thus:

$$L = \sqrt{\frac{1}{N}\sum_{j=1}^{D}(A_0^j)}, \qquad U = 10$$

where $j = 1, ..., D$. At this point I can define the quantities:

$$\hat{u} = \frac{u}{U}, \quad \hat{A} = \frac{A}{A^0}, \quad \hat{p} = \frac{p}{p_0}, \quad x_\star = \frac{x}{L}, \quad u_\star = \frac{t}{T},$$

where $p_0 = \rho U^2$, $T = \frac{L}{U}$ and $A^0 = L^2$.
Now, all dependent variables take values with $O(1)$ magnitude.
The next step is to standardize $x_\star$ and $t_\star$ in order to achieve better performances during the training of the network.

$$\hat{x}^j = \frac{x^j_\star - \mu^j_{x_\star}}{\sigma^j_{x_\star}}, \qquad \hat{t}^j = \frac{t^j_\star - \mu^j_{t_\star}}{\sigma^j_{t_\star}}$$

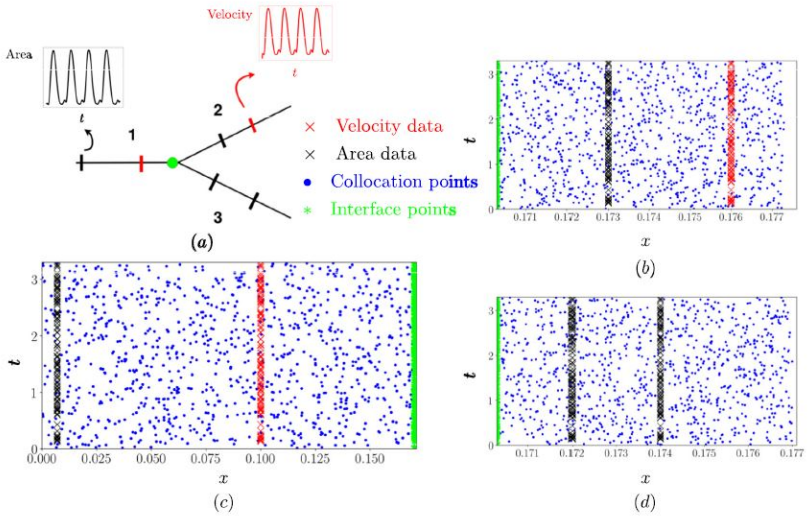

Figure 2: Example of data extraction from a Y-shaped bifurcation

Finally, substituting all these quantities in the physical equations that I defined before, I obtain the equations that I used as constraints in the network training.
For prediction, it is enough to revert equations.

3 Results

I tested the propose method in two applications: a simple **Y-shaped bifurcation** and a complete **upper-arm model**. For the former I considered a prototype arterial network resembling a carotid bifurcation that consists of 3 vessels with 1 bifurcation. For the latter, I take into consideration a complete upper-arm model constituted by 7 vessels with 2 bifurcations. In particular, for this problem I relied on a Python package called *pylsewave* (author: Giorgios Ragkousis) and I used the implemented CFD solver to get input data and test data for the neural networks. All codes are in Python with the Tensorflow package v1.14.

3.1 Y-shaped bifurcation

3.1.1 Arterial Network and parameters

As first example I considered a simple arterial network resembling a prototype carotid bifurcation. The network consists of 3 vessels with 1 bifurcation where each vessel has lengths $L_1 = 170.3$ mm, $L_2 = 7$ mm, $L_3 = 6.7$ mm, respectively, with one bifurcation point at $x_b = 170.3$ mm. At the middle points of each

vessels I have access to $N_u = N_A = 413$ measurements of cross-sectional area and blood velocity generated by a provided Discontinued Galerkin solver for $t \in [0, 3.3]$ s which represents approximately four cardiac cycles with a period of $T = 0.8$ s (see Fig. 2). Notice that I don't assume any input data for pressure for the training, since it's a quantity that I want to predict; whereas for testing I will consider them for comparison. Using these data I inferred the solution at $N_r = 2000$ collocation points, randomly chosen using the Latin hypercube sampling, inside each domain, and $N_b = 1024$ interface points.
Here a table summarizing the main physical parameters of the three vessels:

	Length (cm)	$A_0(m^2)$	$\beta(Pa/m)$	$\rho(Kg/m^3)$	α	K_R
Vessel 1	17.03	1.36e-05	6.97e+07			
Vessel 2	0.7	1.81e-06	5.42e+08	1060	1	0
Vessel 3	0.67	1.36e-05	6.96e+07			

3.1.2 Neural Network setup

I parametrized the solution of this problem using three neural networks, one for each vessel. Each of these networks consists of 7 hidden layers with 100 neurons per layer, followed by hyperbolic tangent activation function. The neural networks are initialized using Xavier initialization. Moreover, I randomly chose batches of $N_{batch} = 1024$ points as input to the Adam optimizer and learning rate $\eta = 10^{-3}$ for the first 92,000 iterations. Consequently, I set $\eta = 10^{-4}$ for the following 40,000 for further minimizing the error and fine-tuning the neural network parameters. To train the complete model it took approximately 12h on my computer equipped with a NVIDIA GeForce 840M.

3.1.3 Comparison of model predictions and reference solution

After the training phase, I extracted the x-coordinate of the three vessels from test data, namely $x_1 = 0.1$ mm, $x_2 = 0.174$ mm and $x_3 = 0.176$ mm respectively, in order to predict all quantities of interest (cross sectional area, blood velocity and pressure). Results are displayed in Fig.3-4-5 Evidently, there is a good match between the solution acquired from Discontinuous Galerkin solver and the one predicted by the proposed physic-informed neural networks model.

Although results are definitively satisfactory, for some vessels the waves shape don't coincide perfectly. This discrepancy is not related to underlying physiological conditions, as for both computational methodologies the same propagation equations are used, but it can be attributed to the representation power of the proposed algorithm. The neural network is probably not expressive enough to capture the small details of the wave-form since there are some difficulties in capturing solutions that exhibit multi-scale features.
Finally, in Fig.6 I showed a comparison of conserved quantities at the bifurcation point. I point out that there is a difficulty in catching peaks for the conservation of momentum, probably linked to the pressure error, since as said this quantity is not provided for training.

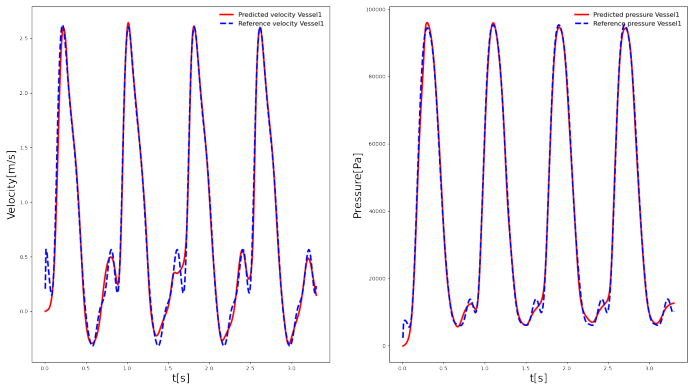

Figure 3: Comparison between model predictions and reference solution for velocity and pressure for vessel 1

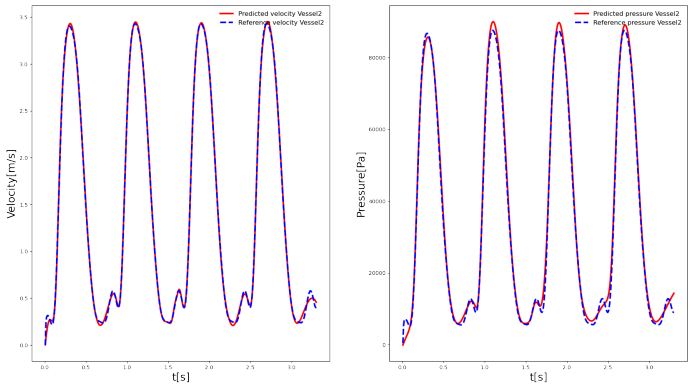

Figure 4: Comparison between model predictions and reference solution for velocity and pressure for vessel 2

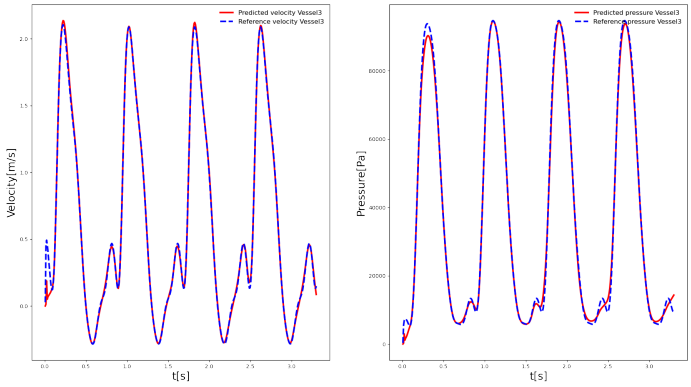

Figure 5: Comparison between model predictions and reference solution for velocity and pressure for vessel 3

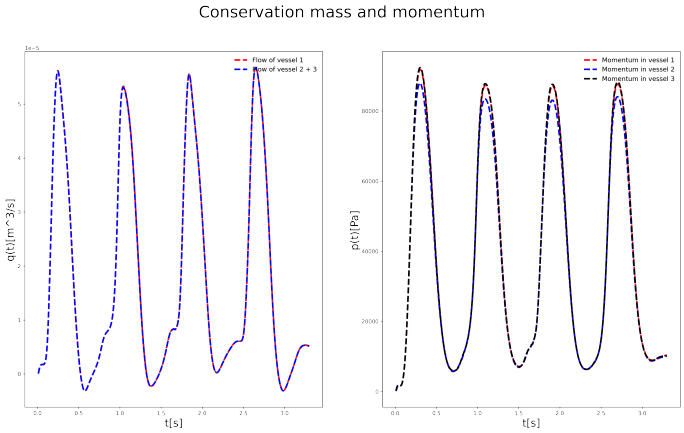

Figure 6: a comparison of conserved quantities at the bifurcation point. The left graphic is the conservation of mass, the right one is the conservation of momentum

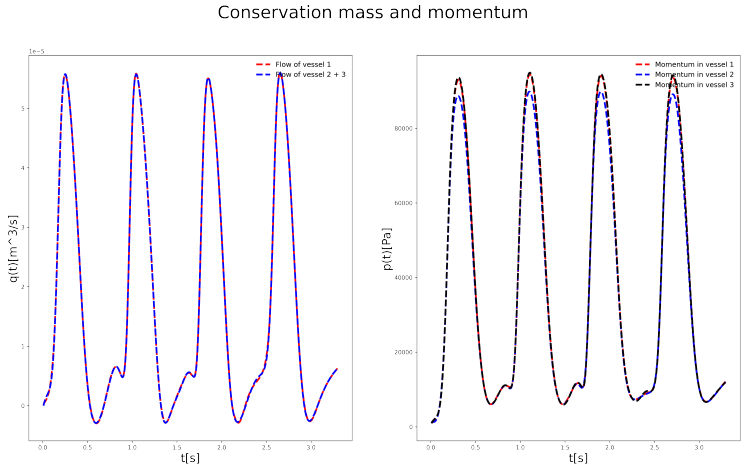

Figure 7: Plot of the continuity conditions at interface with $\beta = 1$

I tried to overcome this problem by adding a higher weight β to the conservation of momentum, namely I trained the neural networks by modifying the loss at the interface as:

$$\mathcal{L}_{\text{interface}} = \mathcal{L}_{\text{mass}} + \beta \mathcal{L}_{\text{momentum}};$$

Due to the time taken for the training of the model, I performed the tests on a lighter (but still decent in performance) neural network composed by 7 hidden layers of 20 nodes each.

As we can see from the plots (Fig.7 - 10), changing the weight attributed to the momentum drastically modify the solution at interface.
However, even weighting the momentum loss much more than the other losses, vessel 2 still has troubles in capturing the pikes of the curve in the momentum plot. Moreover, these changing has a negative impact on the solution, as showed in the following table reporting $\mathcal{L}_2$ errors for the pressure on the test set:

	$\beta = 1$	$\beta = 2$	$\beta = 4$	$\beta = 10$
Vessel 1	0.0441	0.0716	0.3516	0.3807
Vessel 2	0.0453	0.1206	0.4666	0.5739
Vessel 3	0.0420	0.1161	0.4484	0.5471

For completeness, I report the plots of all the losses I obtained during the training (Fig.11).

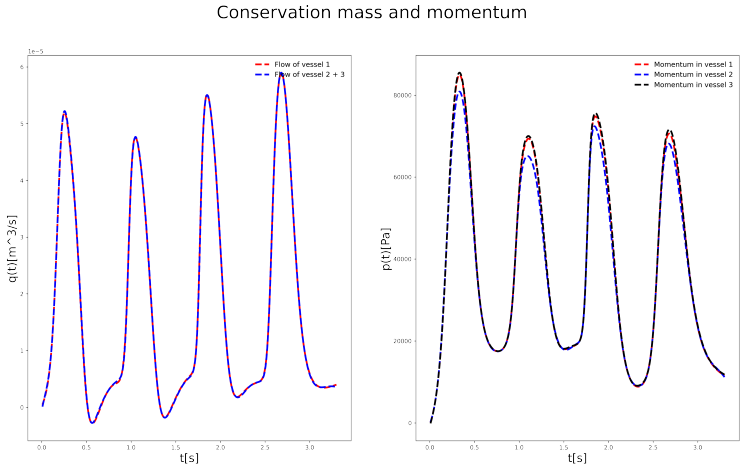

Figure 8: Plot of the continuity conditions at interface with $\beta = 2$

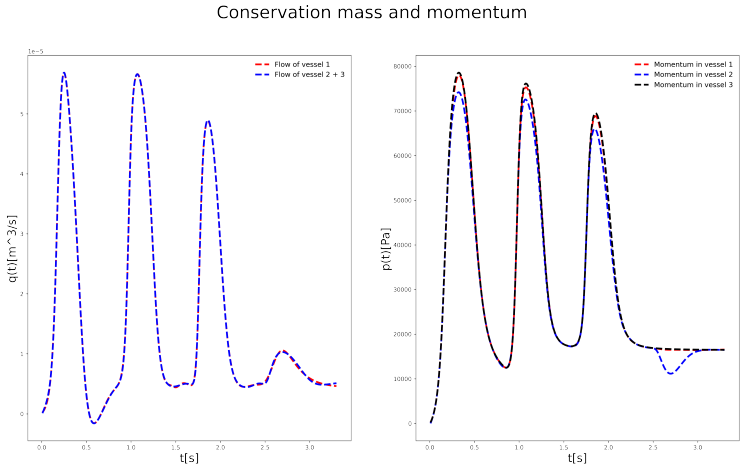

Figure 9: Plot of the continuity conditions at interface with $\beta = 4$

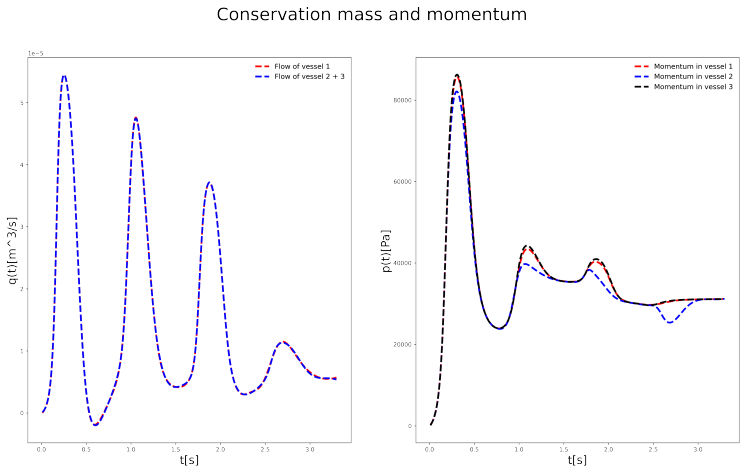

Figure 10: Plot of the continuity conditions at interface with $\beta = 10$

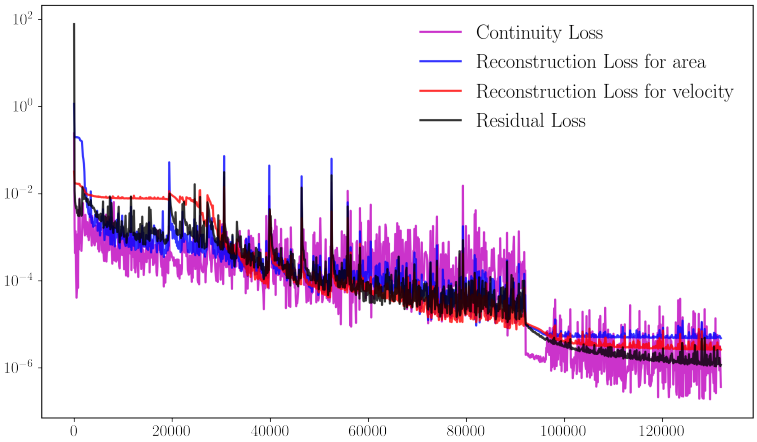

Figure 11: Values of the loss functions versus number of stochastic gradient descent iterations during the training of the physics-informed neural networks.

3.1.4 Systematic studies

In order to test the sensitivity of the proposed methods and quantify their robustness with respect to different hyper-parameter settings, in a first time I computed the $\mathcal{L}_2$ prediction error for 5 different complete neural network initialization using different randomized seeds.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Vessel 1	0.0342	0.0487	0.0573	0.0295	0.0510
Vessel 2	0.0358	0.0456	0.0405	0.0363	0.0686
Vessel 3	0.0314	0.0479	0.0382	0.0184	0.0730

In this framework I computed the $\mathcal{L}_2$ prediction error as:

$$\mathcal{L}_2 = \frac{\sqrt{\sum_{i=1}^{N_f}(p^i_{prediction} - p^i_{measurements})}}{\sqrt{\sum_{i=1}^{N_f}(p^i_{measurements})}} \tag{5}$$

Evidently, results are robust with respect to the neural network initialization as in all cases the stochastic gradient descent training procedure converged to solutions that are close to each other.
Subsequently, I tested the sensitivity of the model with respect to the architecture of the neural networks. In this case, I fix the number of noise-free training data as $N_u = N_A = 413, N_r = 2000$ and $N_b = 1024$ for each vessel. In all cases, I used a hyperbolic tangent activation function and I initialized the neural network weights using Xavier initialization. In the following table I report the relative $\mathcal{L}_2$ prediction error for different fully connected neural network architectures (i.e. different number of layers N_g and number of nodes N_n in each layer).

	$\mathrm{N}_N = 20$	$\mathrm{N}_N = 50$	$\mathrm{N}_N = 100$
$\mathrm{N}_L = 1$	0.7146	0.6508	0.7495
$\mathrm{N}_L = 3$	0.4604	0.1789	0.1267
$\mathrm{N}_L = 5$	0.2840	0.0735	0.0716
$\mathrm{N}_L = 7$	0.0612	0.0330	0.0342

The general trend suggests that as the neural network capacity is increased, I obtain more accurate predictions, which is indicating that the physics-informed constraint on the PDEs residual can effectively regularize the training process and safe-guard against over-fitting.

3.1.5 Noise analysis

In order to test the robustness of the networks, I perturbed initial data with a gaussian noise with zero mean and $\sqrt{\alpha}$ standard deviation, where α is the amount (in percentage) of the given noise. In particular, I tried with 1%, 5% and 10% of noise. I followed the following procedure:

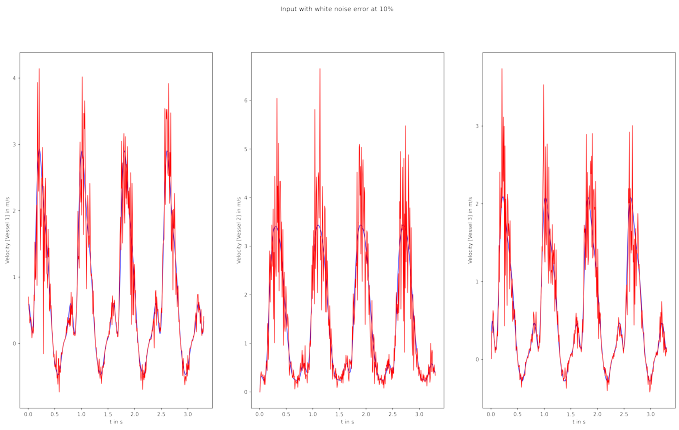

Figure 12: Perturbed velocity measurements with a 10% of noise.

1. Generate a random set of errors from $x \sim N(0, \sqrt{\alpha})$;
2. Generate a new training velocity w as $w = v(1 + x)$;
3. Train the PINN with w.

In a second moment, I considered a normalized noise:

1. Generate a random set of errors from $x \sim N(0, \sqrt{\alpha})$;
2. Compute: $\mathcal{L}_2^{mean} = \sqrt{\frac{\sum v^2}{N}}$ where N is the number of velocity measurements;
3. Generate a new training velocity w as $w = v + x * \mathcal{L}_2^{mean}$;
4. Train the PINN with w.

Both noise strategies brought similar results.
All analysis are conducted on the complete neural network of 7 hidden layers of 100 nodes each. At first, I compared the real pressure with the predicted one and I checked the conservation of mass and momentum by disturbing only the blood velocity data with the different noise percentages considered. As showed in the following figures, the model manifest a certain robustness with respect to noise. I used 10% of noise in all figures below.
In a second step, I tried to perturbe only cross-sectional area and then both area and velocity, but in these two last experiments results are extremely worse than before (noisy cross sectional area data drastically reduce the performance of neural networks).

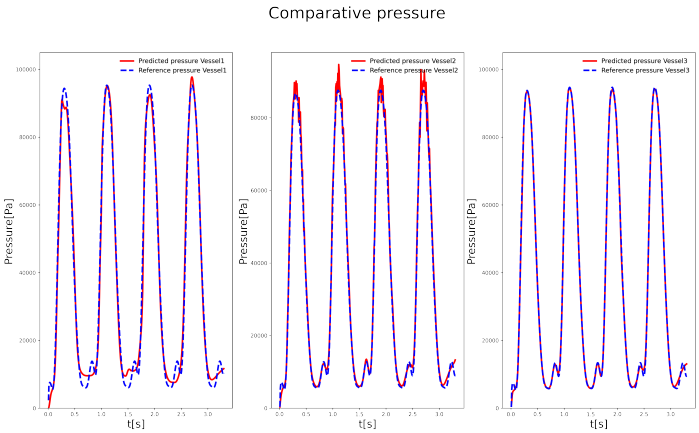

Figure 13: Comparison between model predictions and reference solution for pressure starting from velocity measurements perturbed with a 10% of noise.

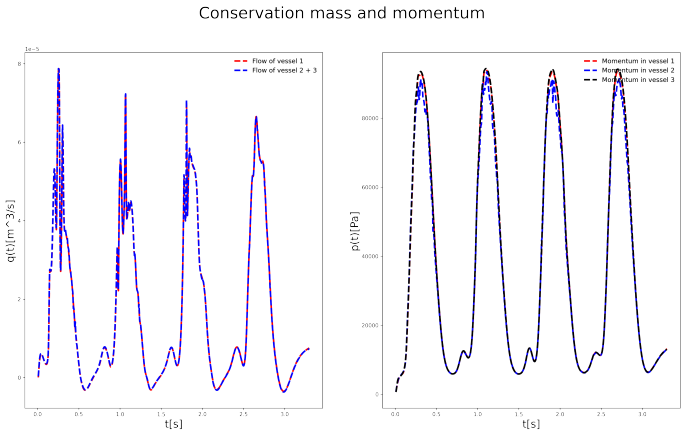

Figure 14: Comparison of conserved quantities at the bifurcation point starting from velocity measurements perturbed with a 10% of noise.

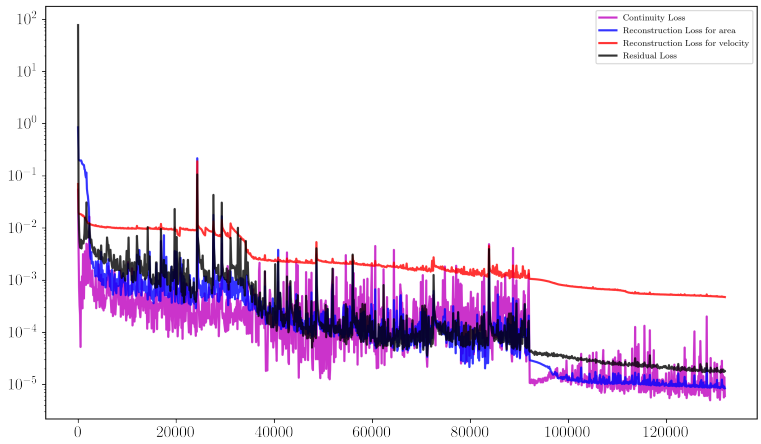

Figure 15: Losses with 10% of noise for velocity data

The $\mathcal{L}_2$ prediction error for these analysis are presented in the following table:

	Velocity			Area			both		
% Noise	1%	5%	10%	1%	5%	10%	1%	5%	10%
Vessel 1	0.03944	0.0466	0.0451	0.4723	0.6273	0.7297	0.3063	0.7008	0.9472
Vessel 2	0.03743	0.0412	0.0623	0.2092	0.6843	0.7349	0.1194	0.4801	0.9353
Vessel 3	0.0364	0.0393	0.0634	0.2164	0.6422	0.7096	0.1263	0.4615	0.9199

Finally, to highlight the role of adding noise to the training of the Network, I report in Fig.15 the behaviour of the loss functions vs the number of iterations when adding a 10% of noise to velocity.

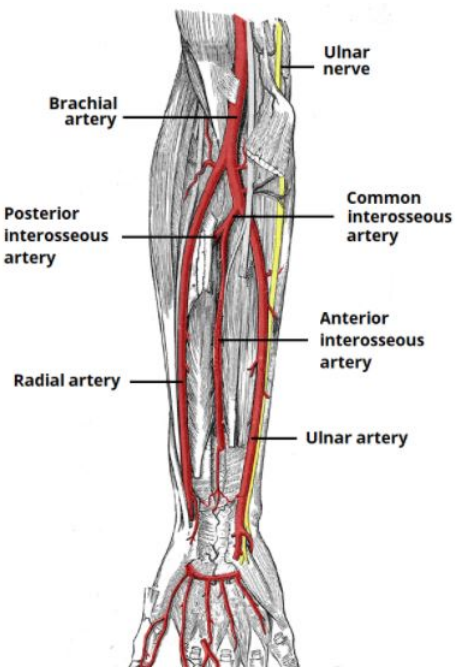

Figure 16: Scheme of an upper extremity model

3.2 Upper arm model

In this section I considered an expansion of previous works, taking into account a real upper extremely model composed by 7 arterial networks. For this application, I used an open source Python package called *pylsewave* (author: Giorgios Ragkousis) and, in particular its CFD solver, for generating data that I used for training my neural networks. Geometrical parameters of this network model can be found in the following table:

No.	Vessel Name	L	R_p	R_d	W_{th}	E
1	Axilliary R	12.00	0.23	0.208	0.067	0.4
2	Brachial R	22.311	0.208	0.183	0.067	0.4
3	R Radial	30.089	0.138	0.138	0.043	0.8
4	R. Ulnar I	2.976	0.141	0.141	0.046	1.6
5	R. Interosseous	1.627	0.096	0.096	0.028	1.6
6	Posterior Interosseous R	23.056	0.068	0.068	0.028	1.6
7	R. Ulnar II	23.926	0.141	0.141	0.046	0.8

where L is the vessel length (in cm), R_p and R_d are the proximal radius and the distal radius respectively (in cm), W_{th} is the wall thickness (in cm) and E is the Young's modulus.

3.2.1 Mathematical problem formulation

The fluid mechanics model that is developed and solved in this package is based on the equations of mass and momentum with density, temperature and viscosity considered as constant. Moreover, it is assumed that all the elements of the model lie on the same height; as a result, the gravitational forces are cancelled. Lastly, the flow is considered to be axisymmetric.
The final PDEs system is quite similar to that one introduced in the section 2, but there are some different aspects that I want to highlight.
Firstly, here the "force" term in Navier-Stokes equation (represented by K_R in the initial system) is not set to zero, but it depends to the velocity profile. In particular, it is assumed that the velocity profile is flat with a boundary layer δ:

$$u_z = \begin{cases} u, & for\ r \leq R - \delta \\ u(R-r)/\delta, & for\ R - \delta < r \leq R \end{cases} \tag{6}$$

In this case, it can be found that the "force" term can be written as:

$$-\frac{2\pi\nu QR}{\delta A}$$

As far as the state equation is concerned, a linear elastic model is used which balances the internal and external forces in radial direction of a surface element on the vessel wall. The pressure-area relation is expressed as:

$$p = \frac{4\sqrt{\pi}E(x)W_{th}}{3A_0}(\sqrt{A} - \sqrt{A_0}) \tag{7}$$

with $E(x)$ and W_{th}, the Elastic modulus and the thickness of the vessel wall, respectively.
However, an exponential empirical model discovered by Olufsen [4] was fitted, expressed as:

$$\frac{E(x)W_{th}}{r_0} = k_1 \exp k_2 r_0 + k_3 \tag{8}$$

Therefore, we can express the elastic modulus with respect to reference diameter r_0 via this empirical relationship. The k vector with the structural parameters can then be obtained with a least square optimisation fit. Olufsen run the optimisation and found that the optimised k vector is as follows:

$$k = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} 2.00 \\ -2.253 \\ 0.0865 \end{bmatrix} \begin{pmatrix} tonnes/s^2mm \\ mm \\ tonnes/s^2mm \end{pmatrix}$$

The above empirical relationship can now be inserted in the pressure-area relationship, trasforming Eq.7 as:

$$p = f(r_0, \mathbf{k}) \left(\sqrt{\frac{A}{A_0}} - 1 \right) \tag{9}$$

with $f(r_0, \mathbf{k}) = \frac{4}{3}(k_1 \exp k_2 r_0 + k_3)$

After these considerations, the system of PDEs that governs the propagation of pulse wave in large arteries is the following:

$$\begin{aligned} &\frac{\partial A}{\partial t} + \frac{\partial Au}{\partial x} = 0 \\ &\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{1}{\rho}\frac{\partial p}{\partial x} + \frac{2\pi\nu u R}{A\delta} = 0 \\ &p = f(r_0, \mathbf{k}) \left(\sqrt{\frac{A}{A_0}} - 1 \right) \end{aligned} \tag{10}$$

with $f(r_0, \mathbf{k}) = \frac{4}{3}(k_1 \exp k_2 r_0 + k_3)$
To deeper analysis and considerations, see the relative document [3].

3.2.2 Neural Network setup

I proceed similarly to the Y-shaped bifurcation. In particular, I considered 7 neural networks consisting in 7 hidden layers with 100 neurons per layer, followed by an hyperbolic tangent activation function. Moreover, I randomly chose batches of $N_{batch} = 1024$ points as input to the Adam optimizer and learning rate $\eta = 10^{-3}$ for the first 50,000 iterations. Consequently, I set $\eta = 10^{-4}$ for the following 15,000 for further minimizing the error and fine-tuning the neural network parameters. For this setup my computer took almost 120h of training.

3.2.3 Coding steps

In the jupyter notebook file called "In_Silico_Upper_Arm_Network.ipynb" there are displayed step by step all pieces of code in order to compute the solution (i.e, velocity, pressure, cross-sectional area and pressure for a certain time interval and certain spatial points) for an upper arm model. I summarize the most relevant steps:
firstly, I have to upload the file "Arterial_Network_ADAN56.txt" containing all vessels parameters shown in the previous table. Then, after defined some simulation constant parameters and variables, I define the mesh which in our case is consisted of different elements representing the arterial segments. Each element can be described with an object instance initiated via the class Vessel. We have to define the inlet boundary condition, where in this case, is an in-vivo interpolated pressure waveform in the aortic arch found in Zambanini [3] and colleague (see "brachial_p_zambanini_invivo.txt"). At each terminal vessel, a

three element Windkessel parameter model has been prescribed. The parameter are defined via an dictionary where each key corresponds to the vessel Id that the lumped model is applied. After defined connectivity and bifurcations, we have to define a name for the simulation run and then I create a callback function that will write results in a user-defined style. The next step is to define the form of the PDE system along with the BCs. In this case, a MacCormack FD solver is used to solve the pulse wave propagation problem in the upper extremity vasculature. In the solver, we have to insert the instances of PDEs, BCs along with the case name and the callback function.
Finally, in this way I get a solution for this model and, in particular, I chose to take training data from the middle point of each vessel and test data from another different spatial point. These data will then be uploaded to the upper arm Python code for both training and testing.

3.2.4 Results

In Fig.17 I compared the real blood velocities and the generated ones by PINNs for all 7 vessels.

As it can be seen in these figures, results are quite satisfactory. In fact, generated velocities seem to follow the shape of the test input data. Instead, as far as the generated pressure is concerned, I didn't manage to get decent results, as one can see from Fig.18 where I displayed the comparative pressure for vessel 3 (the best result I get among all indecent ones).

$\mathcal{L}_2$ errors are all quite high and, as a consequence, conservation of mass and momentum are not acceptable. Indeed, I faced several issues during my works that didn't allow me to get very good results: firstly, adapting the implemented simulated model of the *pylsewave* package to the setup of my neural networks it has been really challenging. Indeed, equations and some parameters used in the solver are quite different from mine. Moreover, I contacted the maintainer of the package (Giorgios Ragkousis) that confirmed me about the non definitive version of the package, due to some oscillations in the solution. Finally, the training for this model was really long due to the huge structure composed by 7 complete neural networks and, for this reason, without the power of a super-computer which can get higher performances, it has been a challenge "debugging" the model and doing some ablation analysis. I hope in future (maybe for my master thesis) to overcome these problems and get better results.

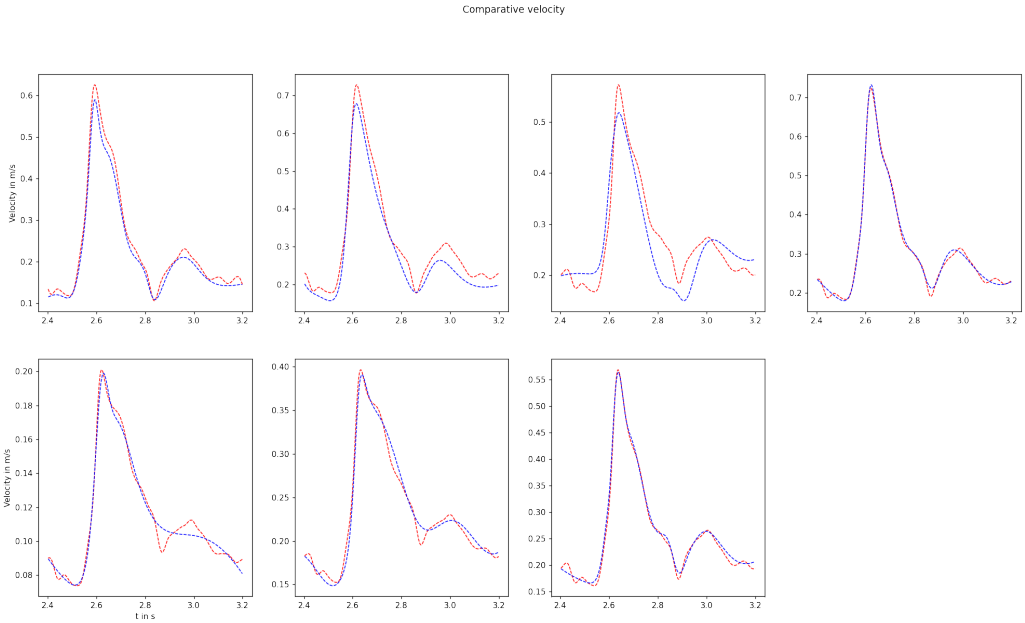

Figure 17: Comparative velocities for all 7 vessels of an upper arm model

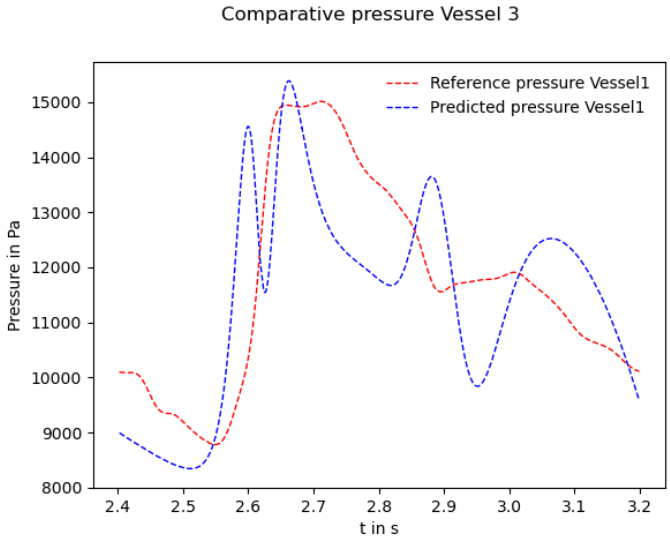

Figure 18: Comparative pressure for vessel 3

4 Conclusion

Advances in physics-informed machine learning provide the connecting link for integrating theoretical models and real-world data. One-dimensional models of blood flow model can be seamlessly synthesized with clinical data to construct deep neural networks that can predict quantities in positions that are different from the ones provided and quantities which cannot be reliably measured in a non-invasive manner (e.g., blood pressure) by complementing the governing laws of fluid flow in compliant arteries with scattered measurements obtained by numerical solvers (but other methods could be employed, like imaging techniques). To facilitate the efficient training of such physics-informed networks, I put forth proper non-dimensionalization and normalization techniques, as well as formulate a composite training objective that enables the consistent propagation of flow information across a network of systemic arteries. For the Y-shaped bifurcation I managed to get excellent results, not only for the blood velocity in positional points that I did not exploit for training, but also for the blood pressure, a more challenging quantity to get in real life and that, for this reason, I generated only by ensuring the solution of the PDEs which describe the physics of the blood inside these vessels. Moreover, due to the lighter structure of the model (at least compared to the more complex model of the upper arm), I had the opportunity to test the robustness of my algorithms for different type of neural networks and with respect to adding noise to input data. For the upper arm model I can not say the same for the reasons that I explained, but for the resources I had, at least I managed to predict good values for blood velocities for all 7 vessels.
In conclusion, PINNs are powerful grey-box methods that nowadays are becoming more and more popular and useful in many applications. The fact that in my examples I didn't have to rely on boundary conditions, meshes or some important data as the pressure, necessary in numerical simulations, make them easily adaptable to different problems. However, due to the ambiguous and complex nature of deep learning algorithms, it is not always simple to fully understand the behavior of the network and there are not still so many solid mathematical foundations available. Moreover, in general these models require a huge computational effort for training and testing, and they are not very practical for further analysis, letting them powerful instruments but with still some limitations.

References

[1] G. Kissas, Y. Yang, E. Hwang, W.R. Witschey, J.A. Detre, P. Perdikaris (2019), *Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks*

[2] M. Raissia, P. Perdikaris, G.E. Karniadakis (2018) *Physics-informed neural networks: A deep learning framework for solving forward and inverse*

problems involving nonlinear partial differential equations

[3] https://pylsewave.readthedocs.io/en/latest/sphinx-rootdir/._pyw_doc001.html

[4] A. S. Olufsen (1998), *Modeling the Arterial System With Reference to an Anesthesia Simulator*

[5] A. Zambanini, S. L. Cunningham, K. H. Parker, A. W. Khir, S. A. M. Thom and A. D. Hughes (2005), *Wave-Energy Patterns in Carotid, Brachial, and Radial Arteries: a Noninvasive Approach Using Wave-Intensity Analysis, American Journal of Physiology - Heart and Circulatory Physiology*

Software used: Python, Tensorflow v1.14, pylsewave v1.0.2