PHYSICS-INFORMED NEURAL NETWORKS: SMALL WAVE SIMULATION

Tiago Silvério (s222963), Luís Freire (s233483), Jacob Schlander (s184031), Bartholomeus Pepping (s223301)

ABSTRACT

This report was written as part of an exploration of the applications of physics-informed neural networks (PINNs) in approximating and adapting to known physical problems. The primary focus was on understanding how these deep learning models could effectively address such problems. PINNs were particularly emphasized due to their capability to achieve accurate approximations while requiring relatively minimal datasets.

To demonstrate their efficacy in approximating a small-scale wave problem derived from fluid dynamics, a PINN was specifically formulated and contrasted against a purely data-driven approach using an identical feed-forward neural network architecture to the PINN. The comparison revealed a substantial reduction in the required data points for the PINN compared to the purely data-driven model, coupled with superior performance in yielding more accurate results. Furthermore, the adaptability of the PINN was notably demonstrated when accommodating the introduction of a body into the water scenario.

***Index Terms*—** Physics-informed neural networks, small wave problem, SoftAdapt

1. INTRODUCTION

In recent years there has been an explosive growth in the amount of data-availability across many separate scientific fields such as image recognition, genomics and cognitive science. This growth in data as well as in computing power has allowed for great progress in the field of machine learning. In other scientific domains involving more complex systems of a biological, physical or mechanical nature data acquisition is still prohibitively expensive. Conclusions in these fields have to be made on limited grounds of partial information. Network architectures such as deep, recurrent or convolutional neural networks fail at providing robust results and guarantees of convergence.

A response to the problem of this small data regime is the application of physics-informed neural networks. The idea behind networks of this type is that we should take into account the vast amount of prior knowledge about the physical laws governing these systems. This would act as a constraint on the solution space, thus amplifying the information of the available data and steering the network toward the solution quicker as well as generalizing more effectively with only a few training examples.

Setting up PINNs we utilize the ability of deep neural networks to universally approximate any function. This allows us to tackle nonlinear problems without having to resort to prior assumptions, linearization or local time stepping. We also exploit automatic differentiation to differentiate the neural network with respect to the network inputs and model parameters. Doing this we obtain a PINN, that is constrained to respect symmetries, invariances or conservation principles.[1]

2. PROBLEM DESCRIPTION

For our studies of PINNs we will apply it to the small wave problem a depiction, which can be seen illustrated in figure 1.

This specific scenario from fluid mechanics is a body of water with small waves at the surface. It is assumed that the amplitude of the wave is significantly smaller than the other behaviours governing the model, and can therefore be physically approximated with the following boundary and fluid conditions:

$$\Phi = \phi, z = 0, \quad \forall 0 \leq x \leq L \tag{1}$$

$$\delta_{xx}\Phi + \delta_{zz}\Phi = 0, \forall z \in (-h, 0) x \in (0, L) \tag{2}$$

$$\begin{pmatrix} n_x \\ n_z \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_z \end{pmatrix} \Phi = 0, \forall [x, z] \in \Omega \tag{3}$$

- Equation (1) denotes the free surface condition at the surface of the fluid.
- Equation (2) the partial differential equation (PDE) that governs the behaviour of the fluid within the fluid itself.
- Equation (3) denotes the no-flux boundary condition at the bottom of the system.

Moreover, there exists an analytical solution governing the system (equation (4)). This will allow us to evaluate the performance of our PINN.

$$\phi = -\frac{Hc}{2} \frac{\cosh\left(k(z+h)\right)}{\sinh kh} \sin \omega t - kx \tag{4}$$

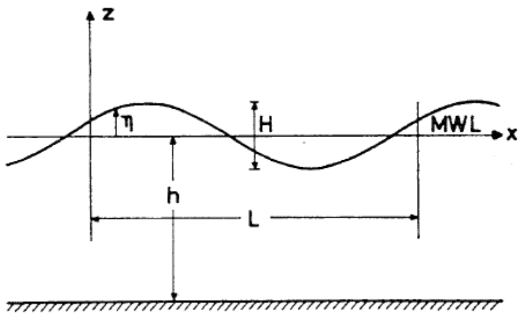


Fig. 1. Sketch of the problem from [2].

3. MODEL

3.1. Architecture

This study examines a multi-layer feed forward PINN using linear layers. FNNs, renowned for their hierarchical architecture, operate by forwarding input data through interconnected layers devoid of feedback loops.
Linear layers are fully connected. Functionally, these layers execute linear transformations on input tensors by employing matrix multiplication with adjustable weights and a bias term addition after which a sin activation function is utilized. The model architecture can be seen in figure 2.
The model is optimized by the Adam optimizer, an optimizer that adjust the learning rate based on the averages of first and second moments as the training progresses. The loss is calculated by smooth L1 loss function. This loss function utilizes a β-threshold to apply the mean squared error if smaller, and the mean absolute error function if the loss is greater than the threshold. The model output of the model is the scalar velocity potential. This is a concept used in fluid dynamics that apply to irrational flows. With the assumption that our fluid lacks vorticity, this allows one to represent the velocity as the gradient of the velocity potential.

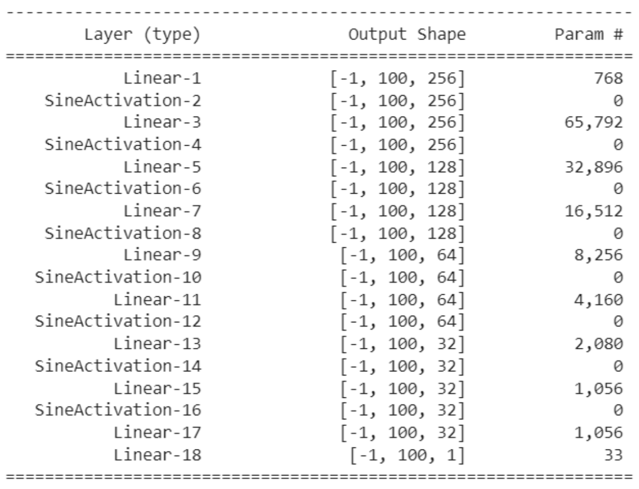
3.2. Model conditions

In the implementation of conditions within our Physics-Informed Neural Network, a meticulous process was undertaken to ensure the model accurately captured the physics of the problem . The essence of this process lies in the generation of data points within each condition domain, maintaining consistency with the same data size.

For each condition, the model's performance was evaluated by calculating the loss specific to that condition. This allowed us to inspect how well the PINN adhered to each governing equation.

To better understand how this loss computation is done we explain how each condition works:

1. *Free surface Loss*: For assessing the model's adherence to the free surface behavior (1), the model evaluates the output at points within this condition domain (z=0 , $\forall 0 \leq x \leq L$) . The average difference between these predictions and the corresponding analytical solution results (equation (4)) is calculated, constituting the *Free Surface Loss*.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1             [-1, 100, 256]             768
    SineActivation-2             [-1, 100, 256]               0
            Linear-3             [-1, 100, 256]          65,792
    SineActivation-4             [-1, 100, 256]               0
            Linear-5             [-1, 100, 128]          32,896
    SineActivation-6             [-1, 100, 128]               0
            Linear-7             [-1, 100, 128]          16,512
    SineActivation-8             [-1, 100, 128]               0
            Linear-9              [-1, 100, 64]           8,256
   SineActivation-10              [-1, 100, 64]               0
           Linear-11              [-1, 100, 64]           4,160
   SineActivation-12              [-1, 100, 64]               0
           Linear-13              [-1, 100, 32]           2,080
   SineActivation-14              [-1, 100, 32]               0
           Linear-15              [-1, 100, 32]           1,056
   SineActivation-16              [-1, 100, 32]               0
           Linear-17              [-1, 100, 32]           1,056
           Linear-18               [-1, 100, 1]              33
================================================================
```

Fig. 2. Architecture of the utilized model.

2. *PDE Loss*: The PDE loss is determined by applying the conditions specified in equation (2) across the entire generated data domain. The model calculates the average result, encapsulating the loss attributed to this condition.

3. *Bottom velocity Loss*: Complying with equation (3), the model computes the velocity in the vertical component at the bottom of the domain (z=-h). This is possible computing the gradient of the model in z direction ($\delta_z \Phi$) at the data points generated at this condition domain. The average result contributes to the *Bottom Velocity Loss*.

4. *Periodic Loss*: Ensuring the model accurately represents a periodic wave involves validating its behavior at *x=0* and *x=L* for points at the same depth. Data within these domains is generated, and the differences between these points at the same depth are computed. The average of these differences constitutes the Periodic Loss.

To consolidate the performance across all conditions, a weighted sum of losses is computed, where each condition's loss contributed proportionally according to its significance. These weights, representing the importance of each condition, were not static but evolved dynamically during the training process using the SoftAdapt technique. This method is described in the following section.

3.3. Adaptive Weight Loss

Traditional approaches to multi-loss functions often involve assigning equal weights to each loss term, assuming that each loss contributes equally to the overall optimization. However, this simplistic approach may not lead to optimal convergence, especially when losses have different characteristics or magnitudes. To address this, the SoftAdapt algorithm was introduced [3]. The algorithm adjust the weights every j number of epochs by weighting them according to equation (5). This will result in assigning smaller weights to functions closer to their minima when SoftAdapt is called.

$$\alpha_k^i = \frac{f_k^i e^{\beta s_k^i}}{\sum_{l=1}^{n} f_\ell^i e^{\beta s_\ell^i}} \tag{5}$$

Here, f_k^i represents the loss function of condition k at epoch i, s_k^i denotes the recent rate of change of the component loss (f_k^i - f_k^{i-1}), and β is a scaling factor that controls the rate of decay of the exponential term. In our implementation, we have opted β=0.1. The weights are normalized to ensure that they sum to 1, effectively adjusting the influence of each loss term based on its current behavior in the optimization landscape. As a result of this technique, the model gets the following *Total Loss* function:

$$\begin{aligned} L_{Total} = \alpha_1.L_{FreeSurface} + \alpha_2.L_{PDE} + \alpha_3.L_{Periodic} \\ + \alpha_4.L_{BottomNormalVelocity} + \alpha_5.L_{Body} \end{aligned} \tag{6}$$

3.4. Data Synthesis

In our approach to synthesize the data for training, we carefully design the data points to capture the essential features of the problem domain. Specifically, we adopt a strategy that combines uniform and random sampling for different regions of interest, but also make sure that the distribution of points across the different boundaries is equivalent, meaning that each boundary has the same amount of points to ensure the model takes equal attention to the various conditions. These points are then standardized using the StandardScaler from SciKitLearn which transforms numerical features by subtracting their mean and dividing by their standard deviation, ensuring that the transformed features have zero mean and unit variance.

3.4.1. Boundary Conditions

For each boundary condition, data points are uniformly generated along the boundaries, providing a solid foundation for enforcing the physics at these specific locations and applies to all boundary conditions, including the free surface of the wave.

3.4.2. PDE Solution Points

To efficiently solve the PDE across the entire domain, a grid is used to span points across the domain, extending from the bottom ($-h$) to the length of the domain (L). Recognizing the need for a diverse set of points to enhance the model's performance, we go beyond uniform sampling and add an equal number of points that are randomly generated across the domain. This combination of systematic and random points ensures that the network can learn both the structured patterns and intricate variations present in the solution.

3.4.3. Validation Set

For validation purposes, only random points are used. This allows us to assess the model's generalization performance on unseen data points, evaluating its ability to make accurate predictions beyond the training set.

3.5. Purely data-driven model

In order to assess the effectiveness of the PINN a purely data-driven approach is taken using the same feed-forward architecture as the PINN, the same hyper-parameters for training and the same number of data points for training. In the PINN, the only points where the solution is given is on the free surface. The same number of points with their corresponding solutions will be given to the data-driven model. These training points will be evenly distributed throughout the entire domain instead of only being on the surface. This will give the model insights into the dynamics below the surface, as well as on the boundaries of the domain.

3.6. Body inserted into the PINN model

3.6.1. Motivation

Upon achieving a satisfactory approximation to the analytical solution in the free-domain problem, our focus shifted towards challenges with no analytical solutions available. Adding a body to the domain is a good example of challenge with no analytical solution defined. If we restrict our problem to a body with defined conditions (geometry, size and position), it would be possible to calculate an analytical solution, but if any detail related to the body changes, it would not be valid anymore. That's where the PINNs' flexibility can be quite useful as it can easily adapt to intricate geometries and positions. This adaptability positions PINNs as a promising methodology for addressing real-world engineering and scientific challenges.

3.6.2. Required changes

The introduction of a body significantly alters the dynamics our model is designed to approximate, requiring thoughtful adjustments.

To accommodate this, we augmented our model with an additional boundary condition. Considering a body S is inserted into the domain:

$$\begin{pmatrix} n_x \\ n_z \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_z \end{pmatrix} \Phi = 0, \forall [x, z] \in \partial(S) \tag{7}$$

Equation (7) states that the velocity in the normal direction to the boundary of the body is null for all points in the boundary. With the inclusion of this boundary condition, we anticipate that the model will demonstrate an enhanced ability to comprehend and account for the impact of the inserted body, thereby providing a more accurate representation of the complex interactions within the entire domain.

To implement this boundary condition into our model, we generate a set of points evenly distributed along the boundary of the inserted body. For each of these points, we compute the normal velocity component to the boundary according to our model, and subsequently, the loss is calculated as the distance from zero. This process ensures that the model is trained to satisfy the additional constraint imposed by the boundary condition, effectively incorporating the influence of the body into its learning.

Besides this, the *free surface* condition that we have considered before is no longer valid due to the impact of the body insertion. In the free domain solution (no body inserted), the *free surface* condition is the only condition that provides desired output values to the network. As we have explained in 3.6.1, when a body is inserted, there are no analytical functions available, so there is no way to replace the *free surface* condition. To deal with this problem we have tried two different approaches: *Body Model 1* and *Body Model 2*.

3.6.3. Body Model 1

With no *free surface* condition, no conditions are directly forced in the model regarding its output. This is clearly a problem, because training a model while only restrictions to its gradients are applied, doesn't provide good results, because only the variations through the domain are evaluated, and not the output values in any part of the domain. For this reason, we have implemented a model, where the *free surface* condition is still considered, but its dynamic weight α_1 is multiplied by a static small value β <1.Besides this, obviously, the *body boundary* condition is added, resulting in the following *Total Loss* function:

$$\begin{aligned} L_{Total} = \beta.\alpha_1.L_{FreeSurface} + \alpha_2.L_{PDE} + \alpha_3.L_{Periodic} \\ +\alpha_4.L_{BottomNormalVelocity} + \alpha_5.L_{Body} \end{aligned} \tag{8}$$

This way the model is not penalized as much as before when the behaviour in the surface doesn't follow the analytical function, giving a chance for the new condition to influence the surface behaviour. Regarding the architecture we have decided to keep the same as the free domain solver.

3.6.4. Body Model 2

This second model, we have opted to follow a different reasoning. The solution deployed for the free domain is a good approximation to the real and known behaviour of the problem so this model is used as pre-trained model for this approach. So in this alternative, the starting point is a solution for the free domain problem, but then the model is trained disregarding the *free surface* condition and considering the *body boundary* condition :

$$\begin{aligned} L_{Total} = \alpha_1.L_{Body} + \alpha_2.L_{PDE} + \alpha_3.L_{Periodic} \\ +\alpha_4.L_{BottomNormalVelocity} \end{aligned} \tag{9}$$

This approach permits the model to start training from a state which the output values are realistic in the context and allows the *body boundary* condition to influence the free surface behaviour without any penalization if it doesn't behave like the analytical function. The problem is that if the learning rate is too high or if the model is trained for too much time, the characteristics that we are interested in the pre-trained model vanish and this approach becomes useless. For this reason, the training epochs and the learning rate are two crucial factors for the success of this alternative.

4. RESULTS

4.1. Comparison to analytical solution

Comparing the PINN result to the analytical solution in figures 3 and 4 show that the network does approximate the analytical solution reasonably well. Although the wave at the surface of the body of water seems to have slightly more velocity and the opposite happens in the lower end of the domain where the velocity seem to be almost null.

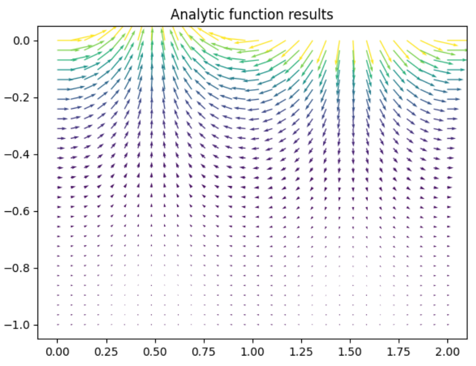


Fig. 3. Results from the analytical model.

4.2. SoftAdapt effect on the loss

In figure 5 the total loss is depicted for the training epochs. In this figure we have compared the model with and without

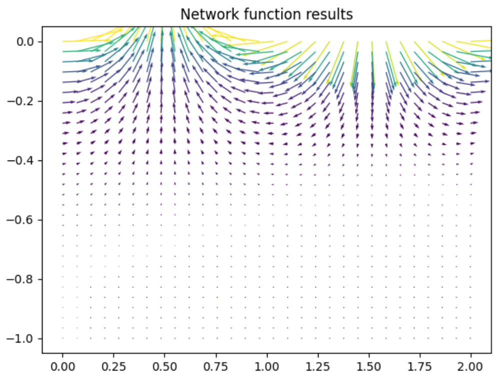


Fig. 4. Resulting flow diagram of the PINN.

SoftAdapt. The training loss reduces significantly compared to the training loss without SoftAdapt. For the testing comparison, the overall loss is slightly reduced compared to the none SoftAdapt version of the model.

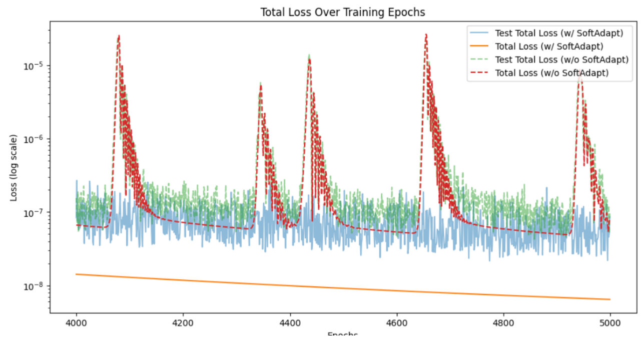


Fig. 5. Architecture of the utilised model.

Figure 6 displays the loss functions utilized in the model with and without SoftAdapt. It can be noted that the individual losses are not necessarily smaller when utilizing SoftAdapt. However, the total loss for the system is minimized.

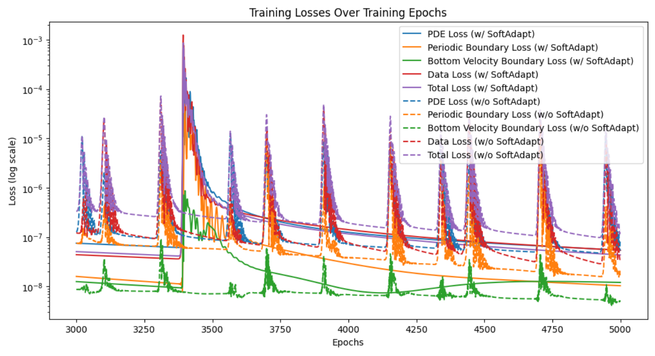


Fig. 6. Training losses comparison w/ and w/o SoftAdapt.

It is also noticeable that the loss is converging to the same value in the various components for SoftAdapt since the weights are dynamically adjusted to ensure that all parts are equally learned by the model, not only the best performing ones whilst without SoftAdapt the boundaries' loss achieve a much lower value, leaving behind the PDE and Data loss.

As for the fitting of the data, the test loss is not far from the training loss and continuously improves over time as can be seen in figure 5 which suggest the model does not overfit but the testing loss is in some components higher than the training loss, specially in the PDE loss. It was observed that when training, the test loss obtained is quite low when training without the random points but much greater compared to the training loss, which suggests that in the future the cut of random points vs uniform points could be changed so that the without increasing the amount of points, the model can still improve its results.

4.3. Comparison with purely data-driven model

Figure 11 shows the resulting gradient field from the purely data-driven model. When compared to the PINN result in figure 4 it can be seen that the data-driven model is clearly inferior given the same number of data-points and training epochs.

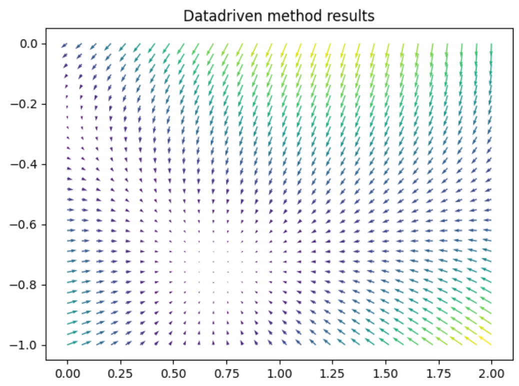


Fig. 7. Results the purely data-driven model.

4.4. PINN with body inserted

4.4.1. Body Model 1

To investigate the impact of the parameter β on the model's performance, we conducted a comprehensive evaluation across three distinct values: $\beta = 0.5$, $\beta = 0.2$, and $\beta = 0.09$. This systematic exploration allows us to discern the varying effects of β and provides insights into its role in shaping the model's behavior and convergence dynamics.

Analyzing the loss plots, it's clear the *periodic boundary loss* is the hardest to control, due to the influence of the body and also the *PDE loss* is quite high compared to other losses

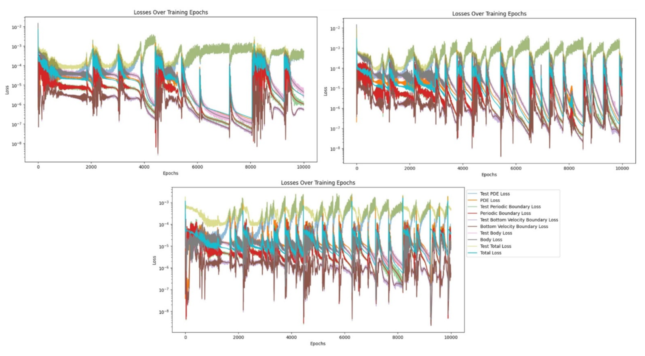


Fig. 8. Losses from each model varying β (0.5 top left, 0.2 top right, 0.09 center)

which is a condition with huge impact in the whole domain behaviour.

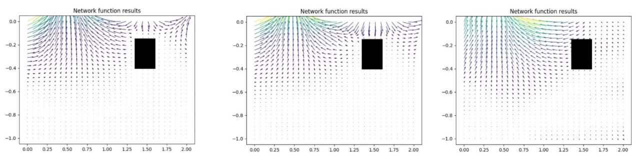


Fig. 9. Results from each model varying β (0.5, 0.2, 0.09 from left to right)

As the value of β decreases, the impact of the body condition becomes more pronounced in the model's predictions closer to the surface. As we expected, a lower β also results in the model lacking clear guidance regarding the desired output throughout the domain. It is evident the velocity in the deeper zones diminishes significantly , which makes sense due to the higher *PDE loss*. This suggests a trade-off between incorporating the body condition effectively and maintaining a comprehensive understanding of the desired behavior across the entire solution domain. From this trade off, we can conclude the configuration that provided better results is.

4.4.2. *Body Model 2*

In this approach, we opted to slightly reduce the learning rate to 10^{-4} compared to the one used in the free domain problem. This modification is made to delve into the interplay between the number of training epochs and the delicate equilibrium between retaining crucial features inherited from the pre-trained model and accommodating the alterations introduced by the body into the domain. Adjusting this parameter allows for a systematic exploration of how the model converges while adapting to the presence of the inserted body. We have evaluated the performance across three different number of epochs: 50,100 and 150.

In the analysis of the velocity field results, due to the low learning rate, the results are quite similar, but we believe these

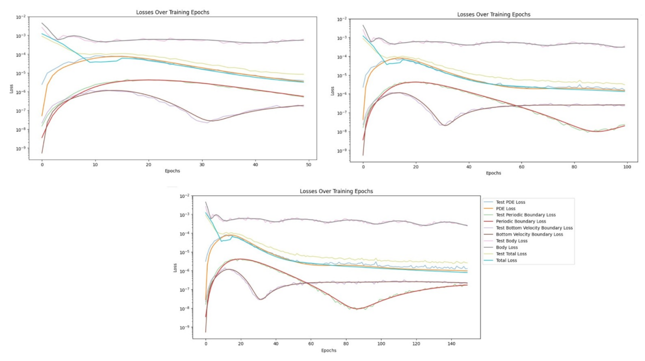


Fig. 10. Losses from each model varying number of epochs

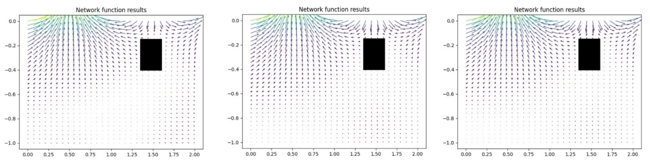


Fig. 11. Results from each model varying number of epochs (50,100 , 150 from left to right)

are good approximations from the real impact of the body. As the number of epochs increases, a similar observation is made to that of decreasing β, wherein the velocity in the deeper regions diminishes significantly. This outcome aligns with our expectations, as mentioned earlier, where the prolonged training causes the model to lose its ability to provide clear guidance on output values.

5. CONCLUSION

In conclusion, the implementation of a PINN using the SoftAdapt algorithm has proven to be a promising approach in simulating the dynamics of small wave problems. Through this study, a comparison with a purely data-driven model showcased the superiority of PINNs in capturing complex physical phenomena while maintaining computational efficiency.

Moreover, the incorporation of an additional body into the system allowed for a more accurate representation of real-world scenarios. This adjustment demonstrated the adaptability of PINNs in accommodating variations and complexities within the problem domain.

The results obtained signify the potential of PINNs equipped with SoftAdapt in solving intricate wave-related problems, paving the way for more robust and efficient computational simulations in various scientific and engineering fields. Further exploration and refinement of this approach could lead to even greater advancements in modeling and understanding complex physical systems.

Github Link

6. REFERENCES

[1] M. Raissi, P. Perdikaris, and G.E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving non-linear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[2] Ivar G. Jonsson Ib A. Svendsen, *Hydrodynamics of coastal regions*, Den Private ingeniørfond, Technical University of Denmark, 1980.

[3] A. Ali Heydari, Craig A. Thompson, and Asif Mehmood, “Softadapt: Techniques for adaptive loss weighting of neural networks with multi-part loss functions,” *CoRR*, vol. abs/1912.12355, 2019.