# Programming Practical: Physics-Informed Neural Networks for inverse problems in hydraulics

H. Boulenc, J. Monnier

January 2024

## Contents

# 1 Neural Networks with Pytorch

Pytorch is a machine learning framework developed by Meta AI since 2016. It is free, open source, and the two main features it allows are :

- Automatic differentiation system via a computational graph.
- Tensor computing with strong acceleration via GPU.

To get a better introduction to the different objects and functionalities that this library offers, a notebook titled "Intro_to_Pytorch.ipynb" is provided. Please have a look and try to experiment a bit with the code to get a grasp of what is possible. The answers to most of your questions are probably inside this notebook, so take the time to understand it well !

# 2 Physical model

For the scope of this Programming Practical, a simplified 1D permanent hydraulics model called the backwater equation model is detailed below. The objective of this algorithm is to infer the spatially-distributed Strickler coefficient, represented by the function $x \mapsto \boldsymbol{K_s}(x) \quad \forall x \in \Omega$.

To solve this inverse problem, an observations dataset $\mathcal{H}_{obs} = \{(h_{obs}^{(i)}, x_{obs}^{(i)})\}_{i=1,\dots,N_{obs}}$ of measurements of the water height $h$ at observation points $x_{obs}^{(i)}$ randomly sampled in the domain $\Omega$ is provided. The $h_{obs}^{(i)}$ are obtained by integrating the backwater equation using a RK4 numerical scheme.

For the backwater equation, the following formulation is used:

$$h'(x) = -\frac{b'(x) + j(\boldsymbol{K_s}; h, x)}{1 - Fr^2(h(x))}, \quad j(\boldsymbol{K_s}; h, x) = \frac{q^2}{\boldsymbol{K_s}^2(x) h(x)^{10/3}}, \; Fr^2(h(x)) = \frac{q^2}{g h(x)^3} \quad \forall x \in \Omega \qquad (1)$$

With $h(x)$, $b(x)$ and $Fr(x)$ respectively the water height, the bathymetry and the Froude number at location $x$. The following constants are also given: $q$ for the flow rate, $g$ for the gravity and $h_0$ for the upstream or downstream water height boundary condition depending on the flow regime :

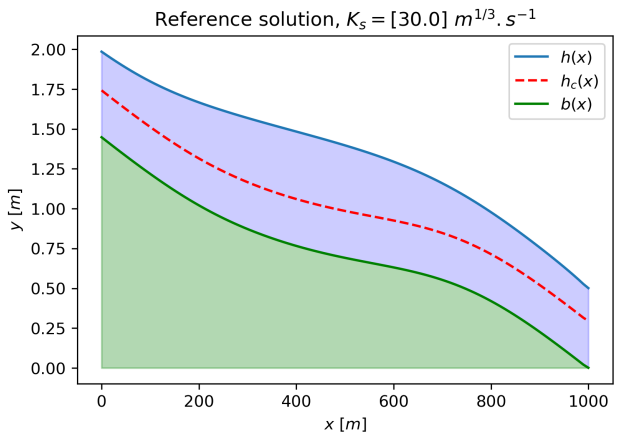


Figure 1: Subcritical regime

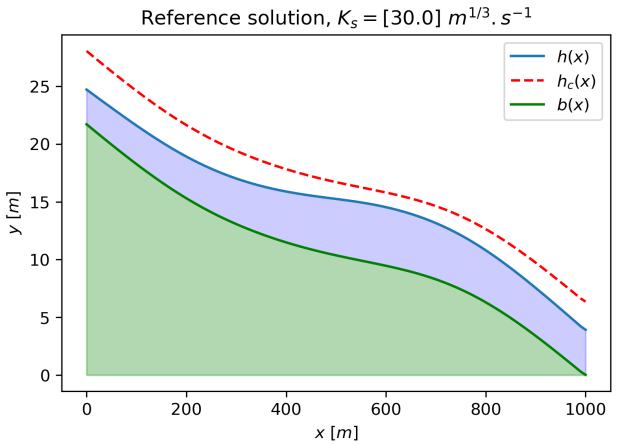


Figure 2: Supercritical regime

# 3 Physics-Informed Neural Networks

## 3.1 Physical residual

From the direct model introduced in Equation 1, the physical residual can be defined as:

$$r(\boldsymbol{K_s}; h)(x) = \dots$$

Once the physical residual is defined, a grid of $N_{col}$ collocation points $\mathcal{X}_{col} = \{x_{col}^{(i)}\}_{i=1,\dots,N_{col}}$ can be sampled in the domain $\Omega$, regularly or not, to minimize the residual on its vertices, called collocation points. Let $\|\cdot\|$ be the norm associated to the euclidean inner product. To embed prior physical knowledge into the training of the neural network, the following physical residual loss function can be minimized:

$$J_{res}(\boldsymbol{K_s}; h) = \dots$$

By using Automatic Differentiation tools (AD), derivatives can be evaluated for a low computational cost and the $J_{res}$ loss function can easily be evaluated at the collocation points. Since $y$ will be approximated by the output of a neural network, it is also important to note that the activation functions $\{\sigma_i\}_{i=1,\dots,d}$ have to be chosen regular enough to be differentiable a sufficient amount of times.

To ensure that the solution satisfies at best the observations $\{h_{obs}^{(i)}\}_{i=1,\dots,N_{obs}}$ distributed over the grid $\mathcal{X}_{obs} = \{x_{obs}^{(i)}\}_{i=1,\dots,N_{obs}}$, a second loss function $J_{obs}$ can be defined :

$$J_{obs}(h) = \dots$$

With $Z$ an observation operator mapping from the physical state space to the observations space, but we won't pay too much attention to this in the current programming practical.

Furthermore, a third loss function $J_{BC}$ is introduced to ensure that the solution will satisfy the boundary condition given with the direct model defined by Equation 1.

$$J_{BC}(h(x_{BC})) = \dots$$

With $x_{BC} = sup(\Omega)$ in subcritical regime, $x_{BC} = inf(\Omega)$ in supercritical regime and $h_{BC}$ given.

To respect both physical constraints and data discrepancy during the training of $\mathcal{N}_{\boldsymbol{\theta}}$, the following total loss function can be minimized :

$$J(\boldsymbol{K_s}; h) = \lambda_{res} J_{res}(\boldsymbol{K_s}; h) \ + \ \lambda_{obs} J_{obs}(h) \ + \ \lambda_{BC} J_{BC}(h(x_{BC}))$$

Where $\lambda_{res}, \lambda_{obs}, \lambda_{BC} \in \mathbb{R}$ are the scalarization factors for the multi-objective optimization problem. For the sake of readability, as the two physics-informed loss functions will always be trained jointly, the $J_{phy}$ loss function is defined as :

$$\lambda_{phy} J_{phy}(\boldsymbol{K_s}; h) = \lambda_{res} J_{res}(\boldsymbol{K_s}; h) \ + \ \lambda_{BC} J_{BC}(h(x_{BC}))$$

With $\lambda_{phy} \in \mathbb{R}$. The total loss function then becomes :

$$J(\boldsymbol{K_s}; h) = \lambda_{phy} J_{phy}(\boldsymbol{K_s}; h) \ + \ \lambda_{obs} J_{obs}(h)$$

## 3.2 Neural Networks with physical constraints

Let's consider a Neural Network $\mathcal{N}_{\boldsymbol{\theta}}$ of the following form :

$$\mathcal{N}_{\boldsymbol{\theta}} \colon x \mapsto \tilde{h}(\boldsymbol{\theta})(x)$$

The training of $\mathcal{N}_{\boldsymbol{\theta}}$ as a Physics-Informed Neural Networks to infer the spatially-distributed Strickler coefficient $\boldsymbol{K_s}(x)$ then refers to the following optimization problem:

$$\boldsymbol{K_s}^*, \boldsymbol{\theta}^* = \underset{\boldsymbol{K_s}, \boldsymbol{\theta}}{argmin} \left( \lambda_{phy} J_{phy}(\boldsymbol{K_s}; \tilde{h}(\boldsymbol{\theta})) + \lambda_{obs} J_{obs}(\tilde{h}(\boldsymbol{\theta}) \right)$$

The architecture of the PINN is represented in the Figure 3.

In the case of the backwater equation, the gradient norm $\|\frac{\partial J}{\partial \boldsymbol{K_s}}\|$ is negligible in comparison to $\|\frac{\partial J}{\partial \boldsymbol{\theta}}\|$ so during the minimization, the physical parameter $\boldsymbol{K_s}$ has no influence and thus gets almost no update. To overcome this issue, an alternating minimization strategy is used :

$$i = 0, \dots, N : \begin{cases} \boldsymbol{\theta}^{(i+1)} & = \underset{\boldsymbol{\theta}}{argmin}\Big(\lambda_{phy} J_{phy}(\boldsymbol{K_s}^{(i)}, \tilde{h}(\boldsymbol{\theta}^{(i)})) + \lambda_{obs} J_{obs}(\tilde{h}(\boldsymbol{\theta}^{(i)}))\Big) \\ \boldsymbol{K_s}^{(i+1)} & = \underset{\boldsymbol{K_s}}{argmin}\Big(J_{res}(\boldsymbol{K_s}, \tilde{h}(\boldsymbol{\theta}^{(i+1)}))\Big) \end{cases}$$

In addition to that, a pre-training where only the $J_{obs}$ functional is minimized is used before the alternating minimization to begin the minimization of $J_{res}$ in a good neighborhood of the physical solution.

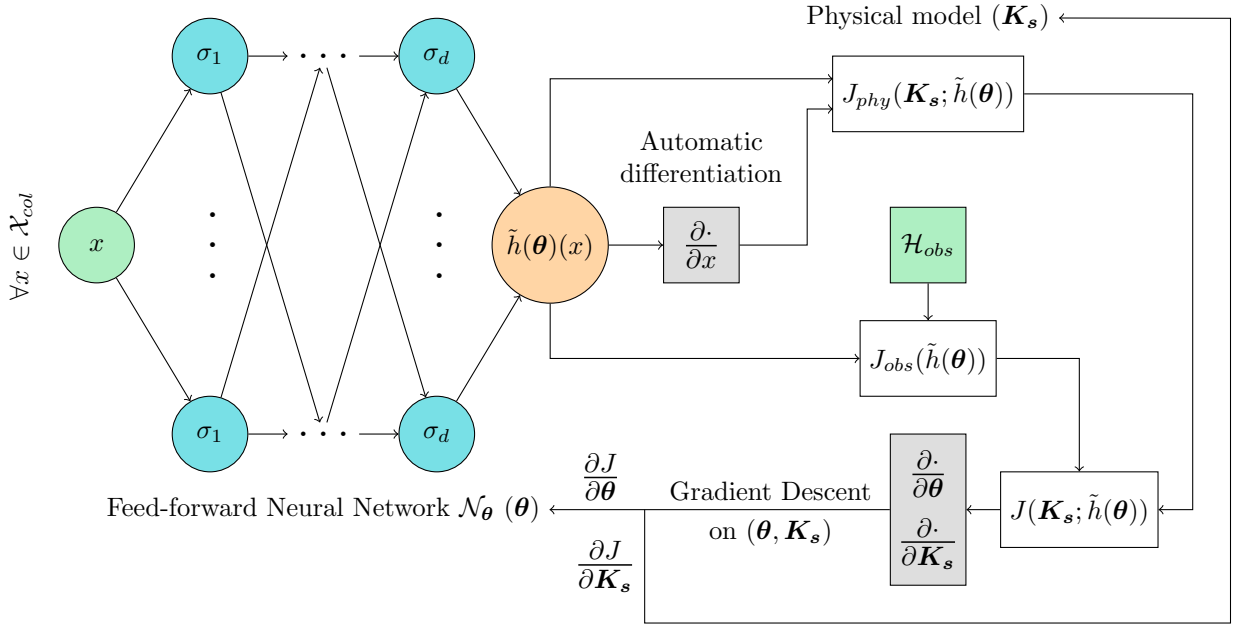


Figure 3: PINN for $\boldsymbol{K_s}$ inference on backwater equation

# 4 Your job

## 4.1 Theoretical study

First of all, you have to define the right loss functions $J_{res}$, $J_{obs}$ and $J_{BC}$ such that when minimizing them, the solution of the Neural Network will satisfy at best the physical constraint defined by the backwater model and the discrepancy with the data.

## 4.2 Build your PINN

Now that you know the expressions of the loss functions, you have to implement them in the code. They are to be implemented in the "Backwater_model.py" file, line 85, 94 and 103 !

## 4.3 Analyze your results

Now your PINN is working fine ! How sensitive is the inference to the different hyperparameters ?

Examples of hyperparameters :

- Number of collocation points $N_{col}$ (can be modified at collocation points generation),
- Number of observation points $N_{obs}$ (can be modified at observation points generation),
- Observations noise (can be modified at observation points generation),
- Number of hidden layers and number of neurons in each hidden layer (can be modified at model initialization),
- Choice of activation function (can be modified in the Class_PINN.py file, line 30),
- Training set size to testing set size ratio (can be modified at collocation points initialization),
- Values of $\lambda_{res}$, $\lambda_{obs}$ and $\lambda_{BC}$ (can be modified in the Class_PINN.py file, line 130 for pre-training and line 149 for alternating minimization),
- Number of iterations of the pre-training and number of iterations on $\boldsymbol{\theta}$ and on $\boldsymbol{K_s}$ during the alternating minimization and number of alternating minimization steps (can be modified at model training),
- Value of $\boldsymbol{K_{s0}}$ at initialization (can be modified at model initialization),
- Method for $\boldsymbol{\theta}_0$ at initialization (can be modified in the Class_PINN.py file, line 46-47, 53-54 and 59),

Choose some hyperparameters in the list (not all !) that you wish to deepen your understanding about and try to evaluate their influence on the results ! To properly measure the influence of one hyperparameter while keeping all the others constants, don't forget to use a seed when creating the model, the collocation points and the observation points ! This can be done in the main.py file, by adding seed = (an integer number).

If you have no idea what to test, you can start by increasing and decreasing the number of collocation points and studying its influence on the inference ! You can do the same with the number of observation points and the noise on the observations.