Miniprojet : Implémentation d'un réseau composite de neurones entraîné sur des données hétérogènes

Baudel Alexandre Boris, Bichon Reynaud Martin et Farhi-Rivasseau Alice,
sous la direction de Bastien Lagneaux et Gurvan Jodin

Ecole normale supérieure de Rennes - Département Mécatronique - 2023

1 Introduction

Le but de cette étude est de tester et d'adapter un type de réseau de neurones afin d'étudier les déplacements de poissons dans un bassin, en temps réel. Les seules données pour déterminer le comportement des poissons sont des données provenant de capteurs sur le bord du bassin (figure 1), ainsi que des données provenant d'une simulation. La mécanique des fluides et les équations de Navier-Stokes seront également exploitées.

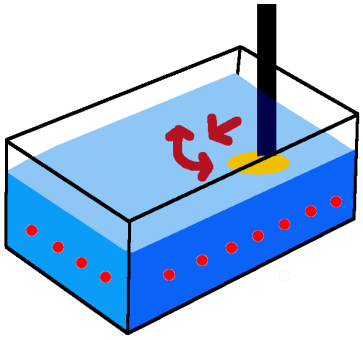

FIGURE 1 – Schématisation du bassin instrumenté et du poisson mécanique.

L'étude a été divisée en trois parties : tout d'abord, le réseau de neurones doit être capable d'extraire une courbe à partir de données hétérogènes. En effet, les données provenant des capteurs sont certaines, mais il y en a très peu. En revanche, les données issues de la simulation sont très nombreuses mais plus éloignées de la réalité. Dans un premier temps, il s'agit donc d'extraire une courbe à partir de ces données hétérogènes. L'équation de la pression sur les bords du bassin est alors déterminée.

Dans un second temps, l'étude se consacre à prendre en compte la variation temporelle, c'est à dire que les données vont varier avec le temps de façon non négligeable.

Cette partie est directement reliée à la dernière partie qui consiste en l'utilisation des équations de la mécanique des fluides afin d'avoir accès au comportement du poisson au milieu du bassin, à partir des valeurs certaines de la pression déterminées précédemment.

Finalement, nous allons aussi étudier, par une autre approche plus classique, le champ de pression et la trajectoire du poisson à l'aide d'une méthode de résolution par éléments finis. L'objectif est de comparer la méthode issue du réseau de neurones et la méthode issue des éléments finis en terme de précision ainsi qu'en coût de calcul.

2 Matériel et méthodes

2.1 Prise en main des réseaux de neurones

L'étude s'est basée en grande partie sur la bibliothèque DeepXDE qui offre de nombreuses structures de réseaux de neurones, comme les réseaux mutlifidelity ou bien les réseau physics informed, qui seront étudiés plus en détail.

Le principe du réseau multifidelity est plutôt simple en apparence : la relation entre les données haute fidélité et basse fidélité est $y_H = \rho(x) * y_L + \delta(x)$ où y_L et y_H sont les données basse et haute fidélité et $\rho(x)$ et $\delta(x)$ sont des coefficients de corrélation. Cependant, ce type de structure ne peut trouver que des relations linéaires entre les deux types de données. Afin de prendre en compte la dimension non-linéaire du problème, la bibliothèque DeepXDE [1] utilise un schéma auto-régressif généralisé $y_H = F(y_L) + \delta(x)$ où $F(y_L)$ est une fonction qui fait correspondre les deux types de données. Cette fonction est décomposée en une partie linéaire et une partie non-linéaire.

Des premiers tests ont été réalisés avec tensorflow, mais il a été jugé peu pratique pour apporter

des modifications. Au final, une version pytorch similaire a été utilisée pour mettre en place le réseau multifidelity[2].

Le réseau de neurones est programmé en accord avec la structure théorique, puisque le réseau est lui-même composé de deux architectures : une architecture low-fidelity et une architecture high-fidelity. Lors du passage dans le réseau, la fonction forward effectue tout d'abord un traitement sur chaque type de données, puis un traitement linéaire entre les deux résultats.

Il est possible d'utiliser la fonction coût ou loss du réseau pour calculer l'erreur lors de la dernière époque. Cette erreur est calculée en sommant les erreurs quadratiques sur les points prédits à hautes et basse fidélités :
$e =< ((y_{predl} - y_l)^2) > + < ((y_{predh} - y_h)^2) >$.
Cela permet en comparant sur différentes fonctions bruitées de voir le niveau de réussite du réseau sur chaque fonction.

Lors de la création du réseau de neurones, le nombre de couches et de neurones par couche est défini. La fonction d'activation joue également un grand rôle. Dans la suite de l'étude, ces paramètres ont parfois été modifiés afin d'obtenir un résultat plus précis. Le nombre d'itérations de l'entraînement est également un paramètre à prendre en compte, puisque une itération peut prendre très longtemps, en particulier pour les réseaux Physics Informed Neural Network (PINN) qui seront étudiés plus tard.

3 Résultats

3.1 Multi Fidelity Neural Network, MFNN

3.1.1 Préparation des set de données

Afin de tester le fonctionnement de ce réseau, des essais ont d'abord été menés sur les set de données de démonstration proposés par les différents auteurs.

Pour le programme en pytorch, un exemple été fournie et testé. Celui-ci ayant rapidement fonctionné, notre propre fonction test a été créée :
$y = (6x - 2)^2 \sin(12x - 4)$.

Créer un set de données bruité a ensuite consisté à modifier cette fonction, de manière à la fois linéaire et non linéaire. Pour pouvoir tester le réseau, il était nécessaire de construire à partir de la fonction de référence les fonctions high-fidelity et low-fidelity. Les points à haute fidélité correspondent aux valeurs des capteurs et sont donc peu nombreux, alors que les points basse fidélité correspondent aux plus nombreux points de simulation. Un programme de génération de ces set de données sur Matlab a donc permis de rassembler la fonction initiale ainsi que les set de données high-fidelity et low-fidelity, pour pouvoir correctement entraîner le réseau.

3.1.2 Entraînement et résultats

La phase d'entraînement sur CPU avec un ordinateur portable dure entre 30 secondes et 1 minute pour 50000 époques, 21 points à basse fidélité et 4 à haute fidélité avec l'architecture du réseau testée.

Pour une première fonction bruitée simple, avec un offset constant égal à 4 (figure 2), le réseau fonctionne correctement (voir figure 3) et trouve la fonction initiale. L'erreur calculée pour lors de la dernière époque vaut ici $e = 4.6 \cdot 10^{-5}$.

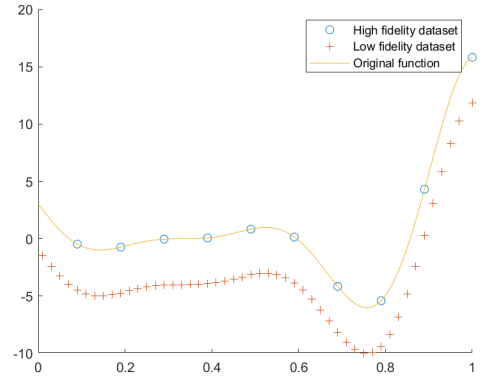

FIGURE 2 – Création du jeu de données sur Matlab.

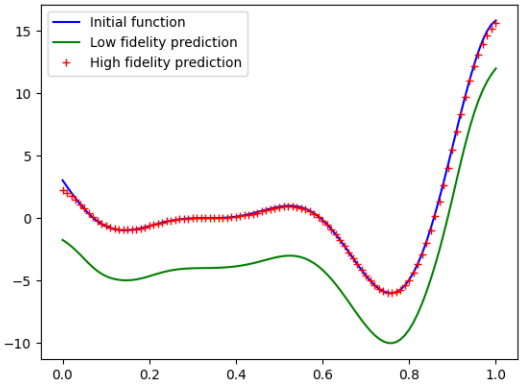

FIGURE 3 – Prédiction à haute fidélité du réseau.

Après s'être assuré que le réseau fonctionnait dans un cas simple, différents essais sur des fonc-

tions à basse fidélité plus bruitées ont été effectués. Un premier essai a d'abord été effectué avec un offset en cosinus : $y_{lowFidelity} = 1.8((6x-2)^2 \sin(12x-4) - 3\cos(1.2x))$.

Le résultat après entraînement figure 4 est satisfaisant et l'erreur n'est que légèrement supérieure à l'essai à offset constant : $e = 4.8 \cdot 10^{-5}$.

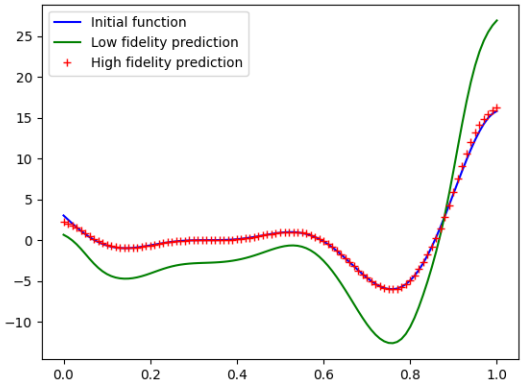

FIGURE 4 – Prédiction du réseau pour un offset en cosinus.

Le réseau fonctionne de manière attendue pour des fonctions peu bruitées, même avec un partie non linéaire.

Il commence a montrer des difficultés à trouver la fonction initiale lorsque les points de basse fidélité sont trop bruités. Il a donc été testé de modifier la partie quadratique des points à basse fidélité (figure 5) : $y_{lowFidelity} = (5.5x-2)^2 \sin(12x-4) - 3$.

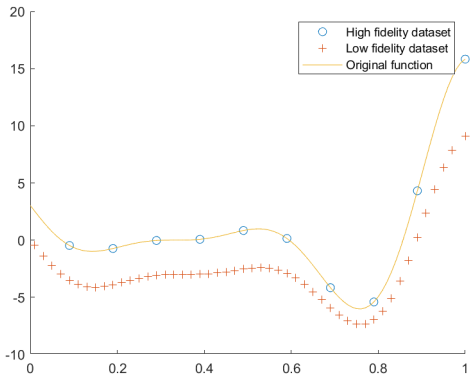

FIGURE 5 – Création du set de données avec un offset non-linéaire.

On trouve pour cet essai une erreur $e = 2 \cdot 10^{-4}$. Elle est bien supérieure à la prédiction de la fonction précédente, ce qui s'observe figure 6 par l'écart plus important entre la prédiction à haute fidélité et la fonction initiale.

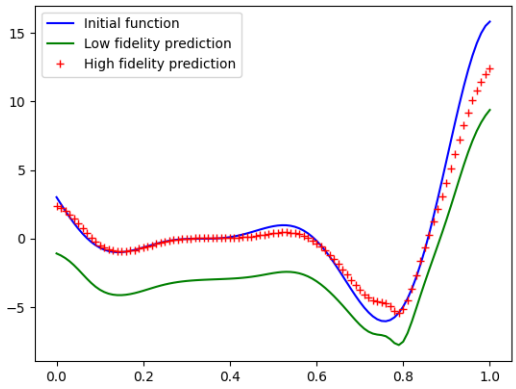

FIGURE 6 – Prédiction pour une partie non linéaire différente.

3.1.3 Essais sur les jeux de données réels

Les essais avec le bassin ayant été commencés, il a été possible de récupérer les valeurs mesurées par un capteur de pression, ainsi que les valeurs de la simulation correspondantes pour un point du bassin.

Ces valeurs étant trop bruitées pour nos tests, il a été tenté de les rendre plus lisses avec une moyenne glissante (figure 7).

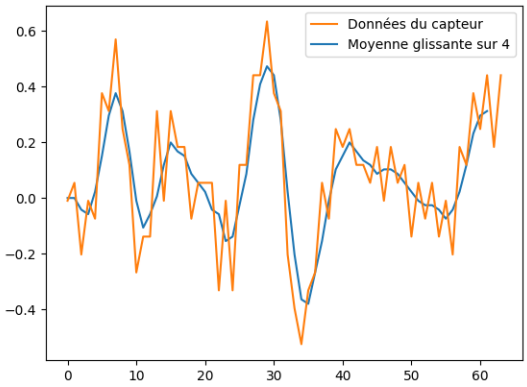

FIGURE 7 – Données capteur moyennées sur 4 points.

Après avoir lancé l'entraînement, on obtient le résultat figure 8, pour une erreur sur $e = 4 \cdot 10^{-4}$. Cette erreur est 8.7 fois plus importante que celle

trouvée pour un offset constant, et le résultat ne semble pas assez satisfaisant pour les étapes suivantes du problème. L'étude de ces étapes a cependant été menée.

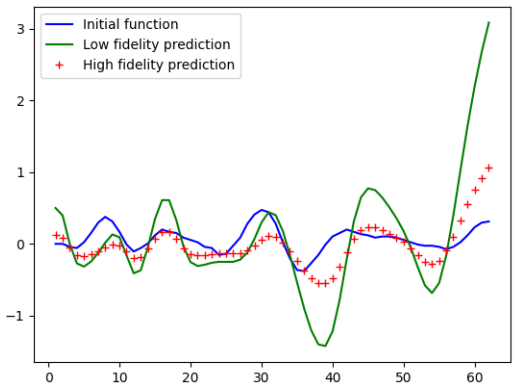

FIGURE 8 – Prédiction du réseau pour pour le jeu de données réel.

3.2 Physics Informed Neural Network

Le but est maintenant de trouver l'équation de la pression en tout point du bassin. L'équation de Navier-Stokes est la suivante :

$$\nabla p = -\rho \frac{D\vec{V}}{Dt} + \rho\vec{g} + \mu\nabla^2\vec{V} \tag{1}$$

Le terme $-\rho\frac{D\vec{V}}{Dt} + \rho\vec{g} + \mu\nabla^2\vec{V}$ fait ici office de terme source, et peut s'exprimer en fonction des données de la simulation. En effet, la vitesse en tout point peut être simulée, et ainsi le gradient ou bien la divergence de la vitesse sont connues.

Tout d'abord, l'étude a essayé d'imposer la condition aux bords. En effet, aux bords, la pression vérifie l'équation trouvée lors de l'application du réseau Multifidelity.

Les données issues du réseau Multifidelity sont interpolées (figure 9) afin d'avoir accès à n'importe quel point de la courbe. Il est possible en diminuant le pas d'interpolation, d'accéder à une approximation plus juste de la prédiction du réseau.

Le réseau de neurones a ensuite besoin de connaître la solution exacte pour s'entraîner. Le poisson est approché par un objet sphérique. Or, l'équation de pression vérifiée par un corps sphérique se déplaçant à une vitesse **U** dans un fluide est de la forme :

$$p = C\eta U\frac{cos\psi}{r^2} = -C\eta div(\frac{\mathbf{U}}{r}) \tag{2}$$

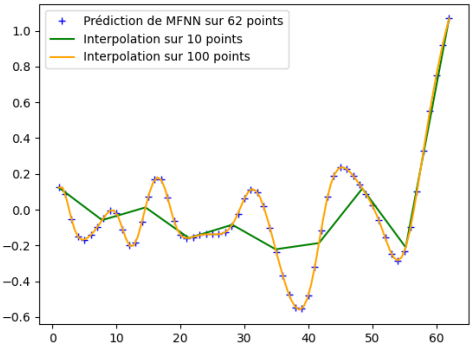

FIGURE 9 – Interpolation de la prédiction de MFNN pour différents pas d'interpolation.

avec ψ l'angle entre **U** et **v** où **v** est la vitesse du fluide, et r la distance entre la sphère et le point d'application de **v**.

Après calculs, la valeur de C est $\frac{3}{2}$, d'où :

$$p = \frac{3}{2}\eta U\frac{cos\psi}{r^2} \tag{3}$$

En utilisant les réseaux PINN de la bibliothèque MFNN [3], il est alors possible d'implémenter nos conditions aux limites et nos équations de propagation dans le réseau de neurones. Le réseau était adapté pour la résolution du champ de vitesse dans un écoulement de Beltrami, et a donc été adapté au problème de l'étude. Les tests se déroulent dans un cube de côté 1m.

La figure 10 présente la pression prédite par le réseau pour un écoulement de Beltrami. L'erreur du réseau est de l'ordre de 10^{-3} pour 2000 itérations, ce qui est satisfaisant.

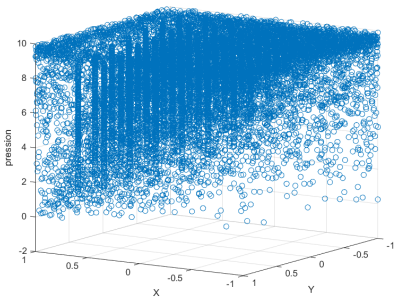

FIGURE 10 – Pression dans le bassin lors d'un écoulement de Beltrami.

Le code est ensuite adapté pour le problème étudié. Le poisson est modélisé par une sphère se déplaçant à 5m/s et ayant un rayon de 20cm. Le champ de vitesse modélisé est présenté en figure 11, et la pression est présentée en figure 12. Le champ de vitesse représente bien la vitesse dans le sillage du poisson, puisque les vecteurs sont de manière générale dans la traînée du poisson, et tournent au niveau du poisson, comme attendu. La vitesse a une une valeur plus importante dans le sillage du poisson que sur les côtés, le résultat semble donc cohérent. En ce qui concerne la pression, celle-ci semble être plus basse au niveau du poisson. Les résultats sont cependant difficiles à interpréter du fait du très long temps de calcul pour une itération et donc de l'erreur lors du calcul du réseau. En comparant le résultat de la figure 12 et celui de la figure 10, il est clair que la forme du champ de pression est incomplète. Cela est confirmé par l'erreur qui est de l'ordre de 10^{-1}. Il faudrait avoir des ordinateurs plus performants afin d'assurer la justesse des résultats.

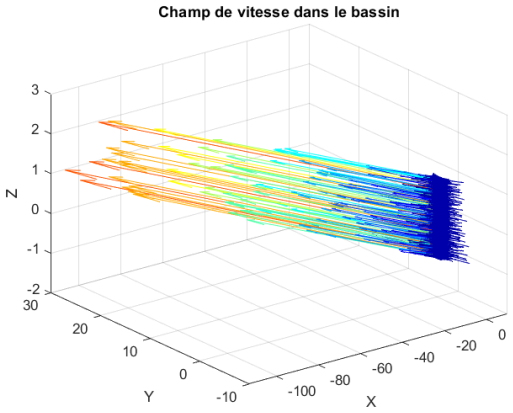

FIGURE 11 – Prédiction de la vitesse.

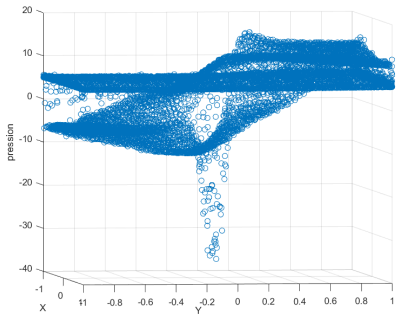

FIGURE 12 – Prédiction de la pression.

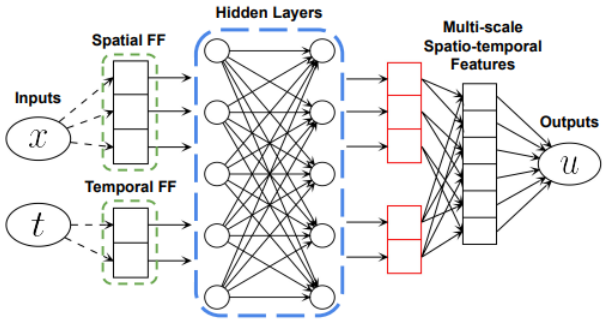

FIGURE 13 – Réseau de neurones et traitements des données

3.2.1 Étude de la dimension temporelle du réseau de neurones

Dans le cadre de l'étude de la dimension temporelle, un pré-traitement sur les données est effectué. En effet, pour pouvoir lier les relations temporelle et spatiale plus aisément, une transformée de Fourier est effectuée sur les données. Après cela, les données spatiales sont entraînées par des réseaux de neurones, puis finalement un traitement multi-échelle est effectué, comme présenté en figure 13).

Dans cette méthode, la reconstitution temporelle est aussi traitée en prélevant les écarts types du PINN (3.2) qui facilitent le traitement de données. Le noyau tangentiel neuronal (NTK) est un outil qui décrit l'évolution du réseau pendant son entraînement par descente de gradient. Il permet de l'étudier à l'aide d'outils théoriques. On dit que la méthode est multi-échelle puisqu'elle est effectuée par plusieurs matrices de noyaux K, dont la valeur des coefficients est donnée figure 14.

$$
\begin{aligned}
(\boldsymbol{K}_{uu})_{ij}(t) &= \Big\langle \frac{d\mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{d\mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \Big\rangle \\
(\boldsymbol{K}_{ur})_{ij}(t) &= \Big\langle \frac{d\mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{d\mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \Big\rangle \\
(\boldsymbol{K}_{rr})_{ij}(t) &= \Big\langle \frac{d\mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{d\mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \Big\rangle.
\end{aligned}
$$

FIGURE 14 – Méthodes de résolution multi-échelle.

L'étude du réseau Physics Informed de DeepXDE [7] renvoie à l'article [6]. L'auteur y indique que les biais et donc les erreurs possibles dans les réseaux de neurones se situent dans les sous-espaces propres du "multi-scale spatio-temporal". Pour pouvoir réduire ces biais, des matrices insérées dans les vecteurs propres sont utilisées. En effet, les méthodes classiques de minimisation d'une fonction coût sont faites avec une seule matrice K de kernel. La mé-

thode présentée en utilise plusieurs et permet de faire l'approximation dans la figure 15 qui permettra de minimiser les biais et la fonction coût.

$$
\begin{aligned}
f(\boldsymbol{X}_{\text{train}}, \boldsymbol{\theta}(t)) - \boldsymbol{Y}_{\text{train}} &= \sum_{i=1}^{N} (f(\boldsymbol{X}_{\text{train}}, \boldsymbol{\theta}(t)) - \boldsymbol{Y}_{\text{train}}, \boldsymbol{q}_i)\boldsymbol{q}_i \\
&= \sum_{i=1}^{N} \boldsymbol{q}_i^T \left(f(\boldsymbol{X}_{\text{train}}, \boldsymbol{\theta}(t)) - \boldsymbol{Y}_{\text{train}}\right) \boldsymbol{q}_i \\
&= \sum_{i=1}^{N} \left(e^{-\lambda_i t} \boldsymbol{q}_i^T \boldsymbol{Y}_{\text{train}}\right) \boldsymbol{q}_i.
\end{aligned}
$$

FIGURE 15 – Simplification pour la minimisation du problème.

Il n'a cependant pas été possible d'implémenter la relation temporelle en raison d'une mécompréhension de certaines méthodes mathématiques du fonctionnement de l'algorithme, et de la difficulté générale de la méthode.

3.3 Modélisation du problème par éléments finis

L'équation à résoudre est l'équation de Navier Stokes (1), présentée plus haut. Les données à relever sont la pression et le champ de vitesse en fonction des coordonnées x et y du bassin. Pour ce faire, un programme informatique déjà existant [4] a été utilisé. Un autre programme permettant la résolution par éléments finis a également été exploité [5]. Le fonctionnement global de la méthode qui est utilisée va être comparé avec la méthode issue du réseau de neurones en termes de temps de calcul et en terme de précision.

Les conditions limites à l'équation de Navier-Stokes sont définies sur les parois du bassin, selon la longueur et la largeur.

$$
\begin{aligned}
u(x, l_y) &= u_N(x) & v(x, l_y) &= 0 \\
u(x, 0) &= u_S(x) & v(x, 0) &= 0 \\
u(0, y) &= 0 & v(0, y) &= v_W(y) \\
u(l_x, y) &= 0 & v(l_x, y) &= v_E(y)
\end{aligned}
$$

La simulation peut être faite en fonction de plusieurs paramètres comme par exemple le nombre de Reynolds pour un fluide turbulent ou laminaire. Ainsi, les équations de Navier Stokes en 2 dimensions deviennent pour la présente étude :

$$
\begin{aligned}
&\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + F \\
&\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \\
&\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = -\rho\left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y}\right)
\end{aligned}
$$

3.3.1 Discrétisation de l'équation de Navier-Stokes

Pour résoudre les équations de Navier-Stokes dans le bassin, les différents termes de l'équation définie dans la figure 16 doivent être discrétisés. Il est ensuite possible d'isoler les (n+1)-ième termes de la vitesse horizontale et verticale, ainsi que de la pression. La discrétisation est faite à un temps n ; i et j représentent les composantes de l'espace x et y.

$$
\begin{aligned}
u_{i,j}^{n+1} = u_{i,j}^n &- u_{i,j}^n \frac{\Delta t}{\Delta x}\left(u_{i,j}^n - u_{i-1,j}^n\right) - v_{i,j}^n \frac{\Delta t}{\Delta y}\left(u_{i,j}^n - u_{i,j-1}^n\right) \\
&- \frac{\Delta t}{\rho 2\Delta x}\left(p_{i+1,j}^n - p_{i-1,j}^n\right) \\
&+ \nu\left(\frac{\Delta t}{\Delta x^2}\left(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n\right) + \frac{\Delta t}{\Delta y^2}\left(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n\right)\right)
\end{aligned}
$$

FIGURE 16 – Discrétisation du champs de vitesse vertical

$$
\begin{aligned}
&\frac{p_{i+1,j}^n - 2p_{i,j}^n + p_{i-1,j}^n}{\Delta x^2} + \frac{p_{i,j+1}^n - 2p_{i,j}^n + p_{i,j-1}^n}{\Delta y^2} = \\
&\rho\left[\frac{1}{\Delta t}\left(\frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y}\right) - \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x}\right. \\
&\left.\frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} - 2\frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y}\frac{v_{i+1,j} - v_{i-1,j}}{2\Delta x} - \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y}\frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y}\right]
\end{aligned}
$$

FIGURE 17 – Discrétisation de la pression

La première modélisation, dont le code est issu de l'article [4], permet de simuler les courants et les champs de pression en l'absence d'objet dans l'eau.

Le second code [5] a été implémenté pour créer un objet dont la position et la vitesse varient au cours du temps, de la même manière qu'un poisson bougerait dans le bassin étudié. Ce code a été adapté pour une représentation en 3D, mais sa complexité est très importante (plusieurs jours de calculs).

3.3.2 Résultats des différentes méthodes

Les deux programmes permettent de visualiser à chaque pas de temps les champs de pression et de vitesse (voir figures 18 et 19). Le champ de pression est représenté sous forme de carte de couleurs et le champ de vitesse sous forme de carte de vecteurs.

Ce sont les trajectoires que les particules suivraient dans l'écoulement si le champ de vitesse était figé.

Les deux simulations ont des temps de calcul similaires en 2 dimensions, et la méthode traditionnelle des éléments finis a permis d'implémenter un poisson entraîné par le courant (figure 18).

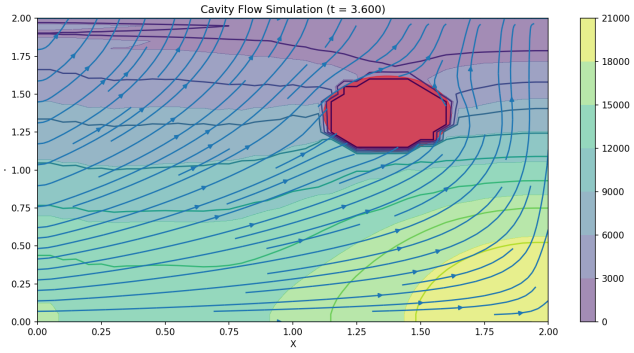

FIGURE 18 – Simulation classique avec un objet se laissant emporter par le courant.

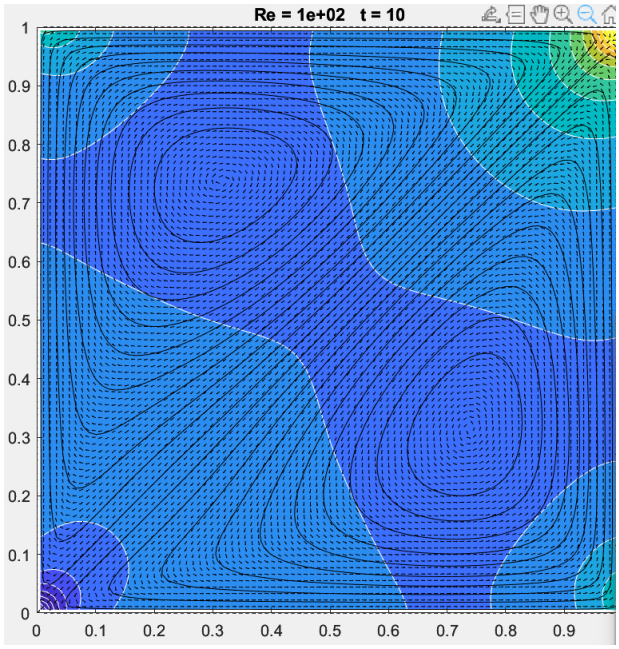

FIGURE 19 – Simulation par la méthode "fast".

4 Conclusion

Afin de répondre au problème de l'étude, chaque partie du réseau global a été étudiée séparément. Les limites de ces méthodes ont parfois pu être observées, notamment lors de leur utilisation avec des données réelles. Elles ont chacune fait l'objet d'outils numériques, physiques et mathématiques différents, en apportant à échelle individuelle des résultats sensés.

La première étape qui consistait à fusionner les données certaines et non certaines n'ayant pas fonctionné de la façon attendue, il n'a donc pas été possible de réunir ces différentes sous-parties en un unique réseau répondant au problème. Cependant, différentes méthodes afin de mettre en place le traitement des données fusionnées ont été étudiées, et certaines semblent exploitables dans le cadre du travail de recherche.

Références

[1] DeepXDE, lululxvi, https ://github.com/lululxvi/deepxde

[2] MultiFidelity Neural Network, MinglangYin, 2019, https ://github.com/MinglangYin/PyTorchTutorial

[3] Exemple de réseau Physics Informed, lululxvi,
https ://github.com/lululxvi/deepxde/blob/master/examples/pinn_forward/Beltrami_flow.py

[4] "A compact and fast Matlab code solving the incompressible Navier-Stokes equations on rectangular domains", Benjamin Seibold, 2008, https ://math.mit.edu/ gs/cse/codes/mit18086_navierstokes.pdf

[5] Navier Stokes par éléments finis, barbagroup, 2019, https ://github.com/barbagroup/CFDPython/blob/master/lessons/14_Step_11.ipynb

[6] "On the eigenvector bias of Fourier feature networks : From regression to solving multi - scale pdes with physics-informed neural networks", Sifan Wang, Hanwen Wang, Paris Perdikaris, 2021, https ://www.sciencedirect.com/science/article/abs/pii/S0045782521002759

[7] "A composite neural network that learns from multi-fidelity data : Application to function approximation and inverse PDE problems", Xuhui Meng, George Em Karniadakis, 2020, https ://www.sciencedirect.com/science/article/pii/S0021999119307260