Vers un simulateur de substitutions pour les écoulements gravitaires

University of Zurich
Rämistrasse 71
8057 Zurich, Suisse

Stage de L3, Mag1

Responsable : Hugo Rousseau
Stagiaire : Baptiste Guilleminot

Du 23 Mai au 12 juillet 2024

Abstract

With the climate change and the important number of infrastructures in the alps, we see emerging a need for larger scale simulations of natural flows. A good example is the destruction of the village of la Bérarde in the Ecrins mountain range in France last June.

The physics of such flows are fairly well known and there exists some numerical methods like the Material Point Method (MPM) which can solve the equations. The issue with such methods is that they are too expensive to use them for monitoring the situation in real time. That's why emulating a neural network simulator could allow, after an expensive training, to lower computation time so it may be possible to anticipate some catastrophic events.

The goal of the internship was to test an option for an emulated simulator : Graph Neural network Simulators (GNS). In order to do that, I will use algorithm for the MPM and GNS which are already coded. I will build a set of simplified data (granular collapse instead of flows) and test different approach to train the neural network. I will then compare the different methods of training and in particular, I will see how they do when I try to scale up the simulation and vary the different parameters using the mean square error on the accelerations.

I'll also try to build a simpler AI algorithm to test an architecture which doesn't need data during training.

Résumé

La forte croissance démographique dans les vallées alpines augmente les risques liés aux mouvements gravitaires (avalanches, glissements de terrain et coulées de débris) pour assurer la sécurité du public et des infrastructures. Avec le réchauffement climatique, la fréquence et l'intensité de ces évènements devraient s'intensifier [14]. Il devient nécessaire de modéliser en temps réel les écoulements naturels lors de situations à risques. Un bon exemple de ces enjeux a été donné en juin dernier avec la destruction du village de la Bérarde dans le massif des Ecrins suite à une succession de laves torrentielles et à la modification du lit de la rivière. Une modélisation efficace des scénarios possibles en amont aurait pu permettre d'envisager les conséquences de telles pluies torrentielles et de réagir au plus vite.

Il y a encore beaucoup à comprendre sur ces écoulements mais certains modèles physiques accompagnées de résolutions numériques telle que la Méthode des Points Matériels (MPM) ont beaucoup gagné en précision ces dernières années [9]. Le défaut de ces méthodes reste leurs temps de calculs. Ceux ci sont encore bien trop importants pour pouvoir utiliser ces méthodes en temps réel. Il serait donc intéressant de pouvoir entraîner un réseau de neurones sur ces simulations afin d'émuler un simulateur qui serait plus efficace.

L'objectif de mon stage est de tester certaines architectures de réseaux de neurones comme les réseaux de neurones en graphes (GNS) [19]. Pour cela, je construirai une base de données de simulations sur une expérience plus simple, l'affaissement d'un tas de sable. J'entraînerai ensuite l'algorithme GNS à partir de cette base de données en utilisant plusieurs méthodes d'entraînement . L'objectif est non seulement de minimiser la fonction de coût mais également de la rendre plus constante sur l'espace des paramètres physiques d'entrée.

Enfin, je testerai une architecture de réseau de neurones plus simple, ne nécessitant pas de simulations pour l'entraînement, pour voir si elle serait suffisante.

Remerciement

Je tiens tout d'abord à remercier **le magistère de physique fondamentale de l'université Paris Saclay**, particulièrement **Patrick Puzo** et **Stéphane Douin** pour m'avoir fait confiance. La possibilité de faire un stage plus long dans le cadre de cette formation m'a permis de mieux en profiter et ainsi d'affiner mon projet d'étude.

Je pense également à toute **l'équipe du projet Coebeli** de l'université de Zurich, qui a cru en mon potentiel et m'a accueilli, en particulier **Martin Luethi**, responsable du projet.

À ce titre, je souhaiterais remercier **Hugo Rousseau**, tuteur de stage, qui m'a non seulement épaulé et conseillé mais c'est aussi démené pour que je puisse réaliser ce stage. Les échanges que nous avons eu furent très enrichissants, ce fut un plaisir de travailler ensemble.

Je tiens également à remercier **Hervé Bellot** pour m'avoir donner des contacts pertinents.

Ce stage m'a permis d'affiner certaines pistes pour bâtir mon projet d'orientation et signe l'aboutissement du cursus de licence ainsi que de la première année de magistère.

Je n'oublie pas non plus **mes proches** qui m'ont sans cesse soutenu dans l'élaboration de mon projet professionnel et m'ont aidé à chaque étape.

Remerciements spéciaux à mon relecteur et correcteur qui a contribué, grâce à ses conseils et recommandations, à l'élaboration et au bon déroulé de ce rapport.

Sommaire

Partie 1

Introduction

Dans le cadre de mon cursus au magistère de physique fondamentale de l'université de Paris-Saclay, j'ai eu l'opportunité de faire un stage durant la fin de l'année scolaire 2023-2024. Ayant deux passions, la physique et la montagne, j'ai eu à coeur de les réunir dans mon sujet de stage. S'orienter vers la géophysique devenait une évidence. Hugo Rousseau m'a proposé un stage au sein de **l'université de Zurich** (UZH) dans le département de géophysique. Mon stage est rattachée au projet COEBELI qui étudie l'évolution du glacier Jakobshavn Isbrae au Groenland.

Même s'il se déroule au sein de ce projet, le sujet est plus large. **L'objectif est de voir si des simulateurs utilisant des réseaux de neurones sont envisageables dans le cas d'écoulements gravitaires.** En particulier, est-ce que les réseaux de neurones graphiques (GNS) entraînés à partir de simulations utilisant la Méthode des Points Matériels (MPM) peuvent reproduire des effondrements granulaires. Je vais également tester un réseau de neurones informé par la physique pour pouvoir comparer les deux méthodes.

L'objectif à terme serait d'avoir un simulateur moins coûteux en terme de calcul pour pouvoir faire des simulation en temps réel. Le travail effectué durant mon stage n'est qu'une première étape dans cette direction pour tester les méthodes sur une situation "de laboratoire".

Partie 2

Description des modèles physiques actuels

Dans cette partie, nous verrons dans un premier temps comment un écoulement réel peut être mis en équations. Nous discuterons en particulier des matériaux élastoplastiques. Nous verrons ensuite une méthode de résolution numérique de ces équations.

2.1 Les équations des milieux continus

Avant de voir les particularités liées aux écoulements réels, rappelons rapidement les grandes équations régissant la mécanique des milieux continus. Deux équations régissent ce domaine. La première correspond à la conservation de la masse. Nous ne nous intéresserons pas à celle-ci car la MPM la résout de manière "automatique" (voir partie 2.3). La deuxième équation correspond à la conservation de la quantité de mouvement. Redérivons rapidement cette équation[1].

Nous nous intéressons à un volume V dans le continuum de matière dont le bord est la surface fermée $S = \partial V$. Le principe fondamental de la dynamique pour ce volume s'écrit :

$$\frac{d}{dt} \iiint_V \rho \vec{u} dV = \sum \vec{F} \tag{2.1}$$

Où $\vec{F}$ sont les forces s'appliquant sur le volume. On sépare ces forces en deux catégories, les forces extérieurs c'est à dire la gravité dans le cas présent, et les forces internes au fluide.

On suppose que les forces internes ne sont ressenties qu'à courtes distances, c'est une hypothèse classique qui est très bien vérifiée pour les fluides classiques et qui est encore plus facile à comprendre pour les écoulements granulaires. En effet, les forces internes sont principalement des forces de contacts. On peut donc réécrire ces forces sous la forme d'un tenseur qui n'est appliqué que sur la surface.

En utilisant le théorème de transport de Reynold, on peut réécrire l'équation 2.1 sous la forme :

$$\iiint_V \frac{d\rho\vec{u}}{dt} dV = \iiint_V \rho \vec{g} dV + \oiint_S \bar{\bar{\sigma}} \cdot d\vec{S} \tag{2.2}$$

Où $\frac{d}{dt} = \frac{\partial}{\partial t} + \vec{u} \cdot \vec{\nabla}$ est la dérivée particulaire.

On pose alors la pression $p = -\frac{1}{3} Tr(\bar{\bar{\sigma}})$ où Tr est la trace et $\bar{\bar{\tau}} = \bar{\bar{\sigma}} + p\bar{\bar{I}}$ sont les contraintes de cisaillements.

Cette équation est vérifiée pour tout volume donc en particulier pour un volume infinitésimale. Pour cela, on utilise le théorème de Green-Ostrogradski pour exprimer l'intégrale sur la surface. On obtient alors

[1] Une démonstration plus complète reprenant en particulier la démonstration du théorème de Reynold peut être trouvée dans [5].

l'équation de conservation de la quantité de mouvement pour un fluide :

$$\frac{d\rho\vec{u}}{dt} = \rho\vec{g} + \vec{\nabla}\cdot\bar{\bar{\sigma}} = \rho\vec{g} - \vec{\nabla}p + \vec{\nabla}\cdot\bar{\bar{\tau}} \tag{2.3}$$

Cette équation est commune à tous les fluides s'écoulant dans un champ de gravitation. Les spécificités du fluide sont alors introduites dans la rhéologie c'est à dire dans l'expression de $\bar{\bar{\tau}}$.

2.2 Le choix d'une rhéologie

Les écoulements réels peuvent prendre différentes formes et sont donc représentés par des rhéologies très différentes. Les grandes catégories sont les rhéologies visqueuses et élastoplastiques. La plupart des modèles actuels sont un mélange des deux. Dans notre cas d'étude, nous nous limiterons à une loi élastoplastique. Pour modéliser l'effondrement d'une colonne de sable (classiquement appelé granular collapse), [7] ont montré qu'une loi plastique pouvait être suffisante. Ici nous utiliserons donc une loi plastique combiné à une loi élastique pour prédire les déformations sous le seuil d'écoulement. C'est donc une loi élasto-plastique. L'ajout d'un modèle élastique permet non seulement de modéliser des propagations d'onde élastique mais aussi de régulariser les équations sous le seuil d'écoulement.

La **loi plastique** choisie est celle proposé par Drucker et Prager [4] pour un milieu non cohésif. Cette loi est caractérisée par la surface d'écoulement définie par $y(q,p) = q - \mu p = 0$ où p est la pression et $q = ||\bar{\bar{\tau}}||$ est la norme du cisaillement définis à la section 2.1. Cette loi est une régularisation de la loi de Mohr-Coulomb et μ est lié à l'angle de friction. Cette surface d'écoulement est représentée sur la figure 2.1.

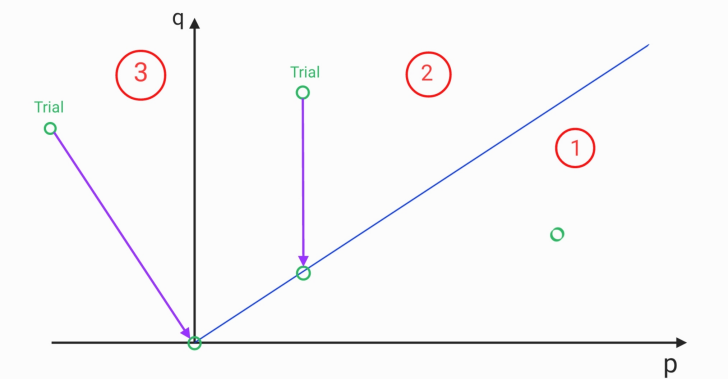

Figure 2.1: Schéma de résolution de la loi plastique de Drucker et Prager sans cohésion ($q = \sqrt{\bar{\bar{\tau}} : \bar{\bar{\tau}}}$).

Afin de **résoudre numériquement** cette loi plastique, on va procéder en deux temps. Il faut d'abord estimer les contraintes en supposant que l'on est dans un cas élastique. Pour cela on peut par exemple utilisé la "finite element method" pour obtenir une solution approché des équations sur une grille. On va alors obtenir un p_{trial} et un q_{trial}. A partir de là, la situation va dépendre de la zone où se trouve ce point sur la figure 2.1 :

- Si le point est dans la **zone 1**, on continue avec ces valeurs pour obtenir la déformation élastique. Il n'y a pas de déformation plastique.
- Si le point est dans la **zone 2**, la déformation plastique est celle nécessaire pour revenir à la surface d'écoulement à pression constante [2], c'est à dire sans déformation volumétrique.
- Si le point est dans la **zone 3**, le matériau se déforme pour revenir sur la pointe de la surface d'écoulement. Il y a alors réduction du volume.

[2] Ce n'est pas la seule solution pour revenir su la surface d'écoulement mais plutôt celle qui correspond aux observations

Les détails de la méthode de calcul de la déformation plastique $\dot{\varepsilon}^P$ à partir de l'essai est décrit dans [1]. A cette déformation on ajoute la déformation élastique $\dot{\varepsilon}^E$ qui est directement liée au tenseur des contraintes $\bar{\bar{\sigma}}$ avec la loi de Saint-Venant et Kirchhoff [1].

Toutes ces considérations 3D nous ont permis de définir une loi proche de la loi de Hook tant que les matériaux restent solides et utilisant la loi de friction de Coulomb pour traduire l'interaction en les grains lors du glissement.

2.3 La méthode des points matériels (MPM)

Maintenant que l'on a une méthode pour calculer les contraintes, il faut choisir une méthode pour résoudre les équations de conservation de la quantité de mouvement donné à la partie 2.1. Pour cela, on choisit la "material point method" (MPM) qui est plus adaptée au grandes déformations. Cette méthode peut être résumée par les 4 étapes de la figure 2.3. Elle explique comment incrémenter d'un pas de temps l'écoulement. Cette suite d'étapes devra ensuite être répétée plusieurs fois pour pouvoir suivre toute l'expérience.

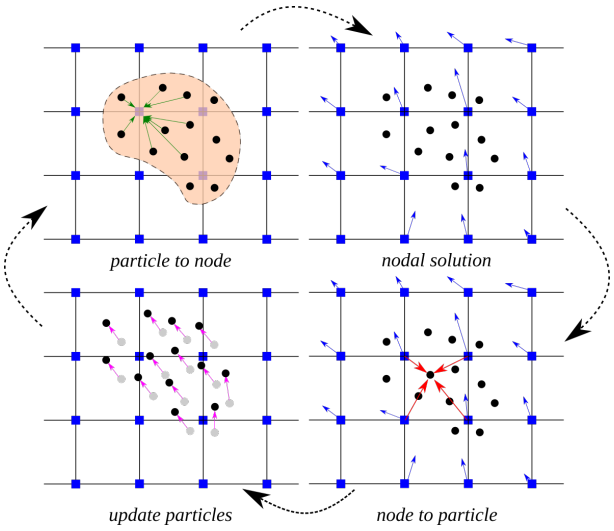

Figure 2.2: Résume les grandes étapes de l'algorithme MPM lors du passage du temps t à $t + dt$.

On part d'un ensemble de particules représentant la situation à un instant t. Chaque particule représente un certain volume de fluide (particule Lagrangienne) et a donc une masse. La conservation de la masse sera automatiquement vérifiée car le nombre de particules reste constant. La vitesse de chaque particule est interpolée sur une grille. Cette interpolation est source d'erreur mais permet de créer la grille à chaque pas de temps et ainsi d'avoir une grille bien régulière. Celle-ci ne vient pas se déformer comme lors de l'utilisation de méthodes en éléments finis (FEM) [15]. On peut ainsi s'intéresser à des déformations plus importantes sans être victime de distorsion de maillage.

L'équation 2.3 peut alors être résolue sur la grille comme dans la FEM. Une fois la grille actualisée, les vitesses sont affectées aux particules en utilisant un modèle FLIP/PIC. Ce modèle est en fait séparé en deux. Avec v_{FLIP} on interpole l'accélération et on en déduit la nouvelle vitesse à l'aide d'un schéma d'Euler alors qu'avec v_{PIC} ce sont directement les nouvelles vitesses qui sont interpolées. On a enfin $v = 0.9v_{FLIP} + 0.1v_{PIC}$.

On a ainsi les nouvelles vitesses, il ne reste plus qu'à mettre à jour les positions des particules. Pour cela, une simple méthode d'Euler sera utilisée. On a alors la situation à l'instant $t + dt$.

2.4 Les limites de cette méthode

La MPM est une méthode très puissante qui est utilisée dans de nombreux cas. Que ce soit pour simuler des avalanches depuis le déclenchement jusqu'à l'arrêt [9] ou des effondrements de terrains comme celui ayant lieu à Brienz en 2023 [10], la méthode a fonctionné à merveille. Elle est aussi utilisée pour modéliser des crevasses [8] ou comprendre le la production d'iceberg par velage au front des glaciers marins [11]. Cette méthode a également été adaptée pour simuler des situations biphasiques.

Le problème pour les modélisations à grande échelles ne vient pas du manque de précision que pourrait engendrer les interpolations. Ce n'est pas non plus la capacité à pouvoir s'adapter à différentes situations physiques, en effet une bonne partie des rhéologies sont déjà codées dans des algorithmes de MPM régulièrement utilisées. Il est tout à fait possible de coder une nouvelle rhéologie sans recoder tout l'algorithme. La MPM est donc adaptable à la plupart des équations du mouvement pour des milieux continus.

Figure 2.3: Exemple de simulation effectué dans [9] suite à des simulations de 1 semaine.

En fait, le défaut de cette méthode se fait sentir au moment de lancer les simulations. Simuler un effondrement granulaire avec suffisamment de précision prend environ une vingtaine de minutes. Des situations plus complexes comme le déclenchement d'avalanche demande d'étudier des phénomènes mésoscopique. Il faut alors quelques dizaines de millions de particules pour être sensible à ces phénomènes comme dans [9], les simulations peuvent prendre jusqu'à une semaine sur 36 CPUs (i9-10980XE). Tout cela malgré une implémentation en C++ très bien optimisée [2].

Ces temps de calculs importants rendent difficilement imaginables l'utilisation de la MPM pour suivre les situations en temps réel ou sur des échelles de temps longues. Ce sont pourtant deux cas très importants. Le premier est nécessaire pour protéger les infrastructures des évènements catastrophiques qui deviennent de plus en plus fréquents. Les simulations longues sont quant à elles nécessaires pour suivre l'évolution des glaciers sur le long terme. Ces simulations vont devenir cruciales, les glaciers étant une source d'eau douce importante dans certaines régions.

Partie 3

Le réseau de neurones graphique (GNS)

Dans cette partie, nous allons discuter de l'utilisation d'une première architecture de réseaux de neurones. Après avoir rapidement expliquer le fonctionnement d'un réseau de neurones, nous explorerons l'architecture du GNS en le comparant à la MPM. Enfin, nous explorerons les avantages et inconvénients observés avec cette méthode.

3.1 Le fonctionnement d'un réseau de neurones

L'objectif d'un réseau de neurones devant simuler une situation est d'approcher au mieux une fonction non linéaire dont la forme explicite est compliquée voir impossible à calculer.

Afin de faire cela, on enchaîne des produits matriciels permettant qu'une valeur de sortie soit dépendante de toutes les entrées. Mais si on en restait à un simple enchaînement de tels produits, la fonction représentée par le réseau serait simplement celle donnée par le produit des différentes matrices. Ce serait donc une fonction linéaire. Or l'objectif est de représenter une fonction non linéaire.

On vient donc rajouter une étape entre chaque produit matriciel. La sortie de la couche précédente passe par une fonction non linéaire appelée **fonction d'activation** avant d'être injecter dans le produit suivant. Il existe un grand nombre de fonctions non linéaires régulièrement utilisées. Une seul condition est vraiment importante : la fonction d'activation doit être suffisamment dérivable pour s'intéresser non seulement à la sortie d'un réseau mais aussi à ces dérivées.

Afin que la fonction approchée soit celle voulue, on entraîne l'algorithme. Pour cela, on compare la sortie du réseau à des valeurs connues. Pour cela, on utilise souvent l'erreur quadratique moyenne qui correspond à la norme de $\mathcal{L}^2$. On cherche alors à minimiser cette erreur appelée **fonction de coût**. Pour cela, on fait remonter le gradient de cette fonction pour comprendre sa dépendance aux différents éléments des matrices du réseau. On vient alors modifier ces éléments dans le sens inverse du gradient. Cette suite d'étapes également schématisée sur la figure 3.1 est répétée de nombreuses fois jusqu'à ce que la fonction de coût soit considérée suffisamment basse.

Pour plus de détails sur le fonctionnement et les différentes solutions techniques, je conseille de consulter le livre de Martin Erdman [13].

3.2 Architecture du GNS

Les Graph Neural Simulator adaptés à la mécanique des milieux continus ont été initialement introduits par Li et al. [20]. Nous nous basons ici sur une architecture plus récente utilisée par Yongjin Choi et Krishna Kumar [19]. Un résumé de l'architecture est proposé dans la figure 3.2. Nous n'allons pas voir les détails

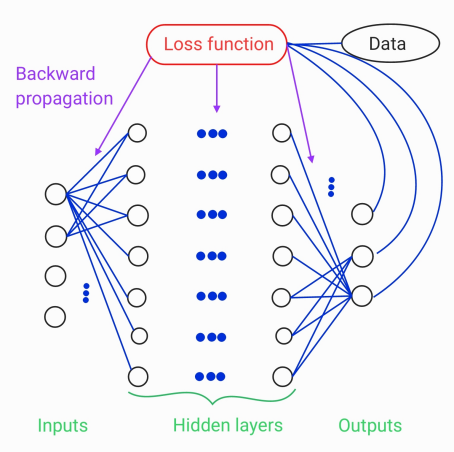

Figure 3.1: Schéma de synthèse sur le cycle d'entraînement d'un réseau de neurones

de l'algorithme mais plutôt chercher à comparer son fonctionnement à celui de la MPM qui sera utilisé pour produire les données.

On peut observer deux **points communs entre la MPM et le GNS**. Le premier est l'utilisation de particules pour suivre le matériau. Cette aspect laisse penser que, tout comme la MPM, le GNS sera relativement bien adapté pour de larges déformations. On sait également que la conservation de la masse sera automatiquement vérifiée. La seule partie à résoudre sera donc la conservation de la quantité de mouvement. Ce point commun semble aussi indiqué que les deux algorithmes seront assez simples à interfacer. En vérité, la MPM sort des fichiers bgeo qui sont très difficiles à lire. Il a donc fallu passer beaucoup de temps pour comprendre comment rendre le processus automatique[1].

On observe aussi que dans les deux cas, la dernière étape est une méthode d'Euler qui permet de calculer le déplacement entre l'instant t et $t + dt$. Cela laisse penser que le GNS pourrait être un bon candidat pour reproduire des simulations MPM.

Les **deux méthodes diffèrent** plutôt sur la manière de calculer l'accélération, c'est à dire sur la manière de faire le bilan des forces. Comme expliqué au paragraphe 2.3, la MPM projette sur une grille les particules et résout alors les équations vues au paragraphes 2.1 et 2.2. Le GNS va plutôt se servir d'un graphe pour encoder les informations liées à la position au niveau des sommets et les interactions sur les arêtes. Afin de calculer les nouvelles accélérations, l'algorithme va utiliser à plusieurs reprises un réseau de neurones (étape "message passing" de la figure 3.2). L'accélération est ensuite décoder et renvoyer aux particules. On se retrouve alors dans la même situation qu'avec la MPM quand les vitesses ont été interpolées de la grille sur les points.

Quels sont alors les avantages de cette nouvelle méthode?

3.3 Des aspects prometteurs

Une des promesses de cette architecture d'intelligence artificielle est d'apprendre la physique cachée derrière le problème et pas juste l'évolution la plus probable du système. En effet, une bonne partie des algorithmes d'IA entraînés sur des fluides sont des réseaux convolutionnels (voir [13] pour comprendre l'architecture) prenant en entrée une grille avec les vitesses et d'autres informations physiques à l'instant t et sortant une grille pour l'instant $t+dt$. Ce qui est appris est alors l'évolution du fluide et non sa physique.

En revanche, le GNS apprend la physique et laisse l'évolution à une méthode plus classique (schéma d'Euler). Cela devrait donc permettre de mieux gérer le fait de changer de situation, en particulier au

[1]La solution trouvée pendant ce stage est assez complexe et demande de rentrer un peu dans le code. Elle est cependant expliquer dans ce dossier GitHub.

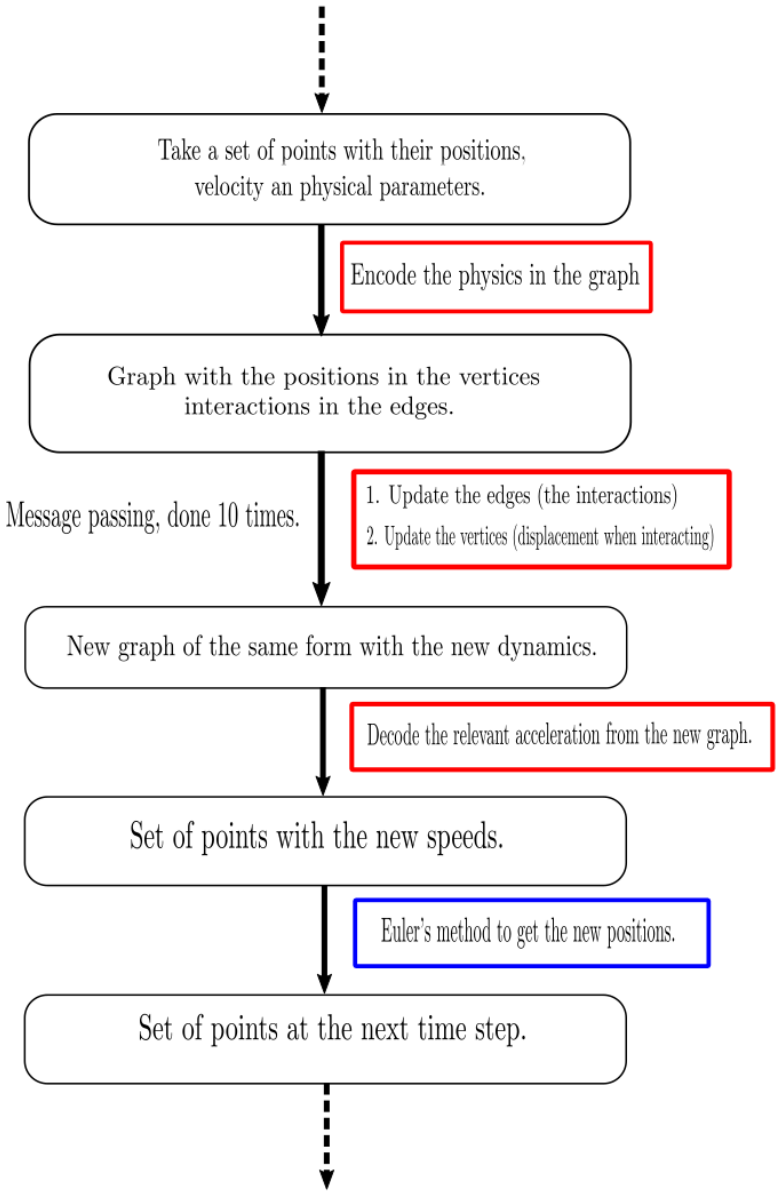

Figure 3.2: Résumé de comment le GNS fonctionne. Les étapes encadrées en rouge sont effectuées à l'aide d'un réseau de neurones. Celles encadrées en bleu sont effectuées avec des méthodes "traditionnelles".

moment de traiter des simulations à plus grandes échelles ou lors de changement important des conditions aux limites. Cet aspect serait particulièrement intéressant pour la généralisation d'un système de prévention.

Je n'aurai malheureusement pas le temps de tester cette hypothèse durant mon stage n'ayant pas eu le temps d'entrainer assez le GNS et n'ayant pas trouvé une implémentation de réseau convolutif satisfaisante pour comparer.

L'autre promesse est une accélération très importante du temps de calcul comparer à un schéma de MPM.

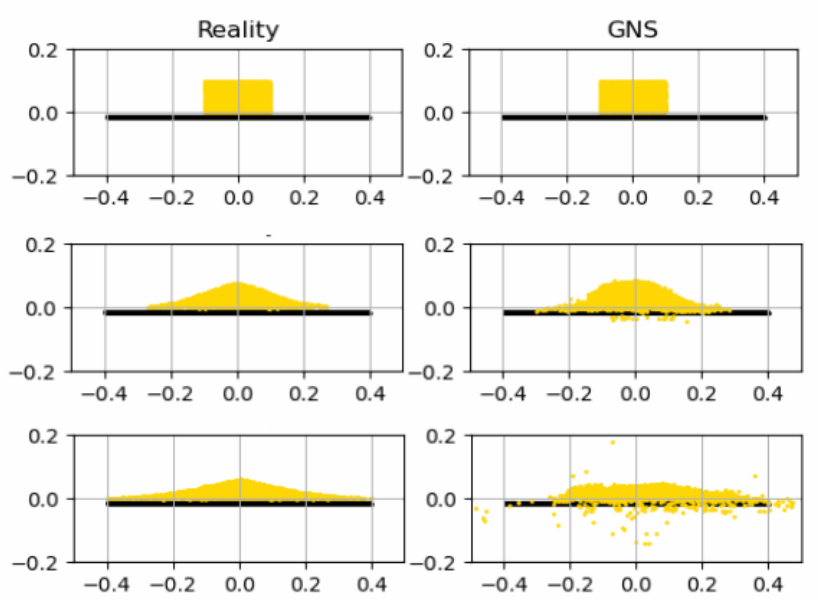

Figure 3.3: Comparaison d'une simulation MPM et d'une simulation GNS après 48h d'entraînement.

Dans leurs articles, Choi et Kumar parlent d'un temps de calcul divisé par plus de 1000. Cela est sûrement vérifié et utile pour de grosses simulations mais avec un GPU de 8Gb, le nombre de particules dépasse difficilement les 10000. Une accélération est donc présente et est perçue à l'utilisation mais n'est pas crucial car les simulations MPM tournent en l'espace de quelques minutes.

De plus, le modèle doit être chargé avant d'être utilisé ce qui rend la différence de temps encore plus réduite. On arrive donc aux limites de l'algorithme.

3.4 Les limites de cette méthode

Le GNS présente des avantages, en particulier la vitesse de calcul. Cette vitesse vient avec l'utilisation de GPU qui ont un espace limité en terme de mémoire vive. En effet, cette mémoire ne peut pas être étendue temporairement comme avec un CPU. Cet espace limité va poser problème car le nombre de particules nécessaire pour avoir une résolution raisonnable sur un écoulement oblige l'utilisation d'un GPU ayant au moins 40Gb de mémoire.

Cet aspect est amplifié par la nécessité de définir les obstacles et surfaces d'écoulements avec des particules. Contrairement à la MPM qui peut mettre les obstacles sur la grille et les traiter séparément, l'implémentation du GNS utilisée demande à ce que les obstacles soient décrits par des particules ayant un paramètre spécifique. Les obstacles, surtout en trois dimensions, prennent donc rapidement un espace important dans le GPU. On se retrouve alors avec des obstacles mal traités à cause du manque de finesse de leur représentation.

Ces problèmes sont clairement visibles sur la figure 3.3 et ne peuvent être résolus que de deux manières :

- Avoir un ordinateur avec un plus gros GPU afin de ne plus être limité par la place sur celui ci.
- Changer l'implémentation du GNS, en particulier le traitement des conditions aux limites. Or l'architecture complexe du code et l'optimisation effectuée sur celui-ci rend difficile toutes modifications par quelqu'un qui n'est pas expert dans le domaine.

Cependant, on observe que les premières images de la figure 3.3 sont relativement fidèles. Je vais donc me baser sur ce constat pour étudier certaines méthodes d'apprentissages.

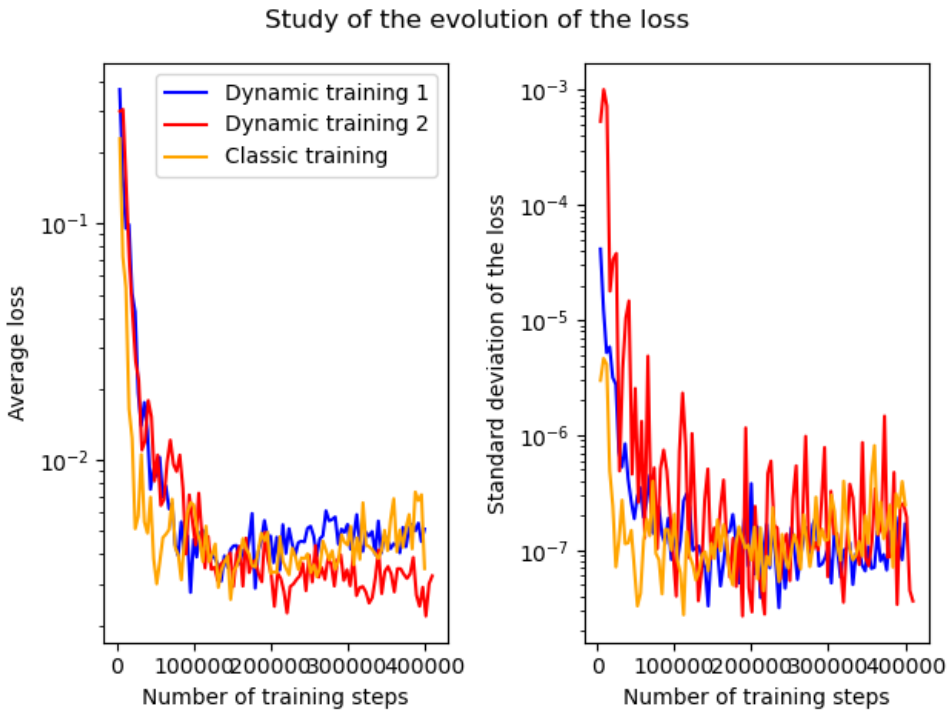

Figure 3.4: Évolution de la fonction de coût et de son écart type au cous de l'entraînement. Dynamic training 1 correspond à la deuxième méthode d'entraînement présenté et Dynamic training 2 à la troisième.

3.5 Études de différentes méthodes d'apprentissages

Malgré le peu de réussite apparent de la méthode, les premiers incréments de temps se rapprochent de la réalitée. Du coup, je poursuis encore un peu dans cette voie. Un point important : on ne veut pas un simulateur qui soit limité à un seul matériau. Justement, l'implémentation de Kumar laisse un paramètre de taille quelconque permettant de préciser les spécificités physiques de la simulation. Je voulais alors voir s'il était possible de modifier l'entraînement afin de mieux prendre en compte ces paramètres et pouvoir éventuellement améliorer les simulations en dehors de la plage vu lors de l'entraînement. Pour cela, je propose trois méthodes :

- D'abord un **entraînement classique**, je sépare les simulations ayant les mêmes paramètres physiques (8 pour chaque jeu de paramètres) et j'entraîne autant chaque paquet de simulations.
- Une seconde solutions est de séparer deux cas. Là où la fonction de coût est faible (en-dessous de la médiane), on utilise que 5 simulations. Par contre, si la fonction de coût est plus élevée, on en utilise 10.[2]
- La troisième variante consiste à entraîner deux fois plus le réseau sur les points où la fonction de coût est faible.

Les 2 dernières méthodes seront dites **dynamiques**.

[2]Les simulations utilisées pour l'entraînement ne sont pas effectuées exactement avec les valeurs physiques données mais avec des valeurs légèrement modifiées aléatoirement. Cela permet de contrecarrer les erreurs systématiques informatiques. Je me sert de cela pour essayer de renforcer l'entrainement.

J'entraîne alors trois fois le GNS en faisant varier l'élasticité et l'angle de friction dans le paramètre supplémentaire et je m'intéresse, entre les différentes simulations, à la valeur moyenne de la fonction de coût et à son écart-type. Les résultats sont donnés dans la figure 3.4. On observe que l'écart type de la fonction de cout à tendance à remonter avec l'entraînement classique au bout d'un certain temps alors qu'il semble plutôt se stabiliser avec les méthodes dynamiques. Mais la figure reste très chaotique rendant une étude plus précise compliquée.

Une remarque supplémentaire sur la valeur moyenne est la suivante : les deux premières versions d'entraînement semblent arriver à un plateau alors que la troisième continue à descendre.

Il semble que cette troisième méthode d'entraînement soit légèrement préférable. De plus, elle demande moins de simulations en entrée. Ce résultat est cependant très discutable car il faudrait moyenner chaque courbe de la figure 3.4 sur plusieurs entraînements pour avoir des résultats solides. En effet, les réseaux de neurones sont initialisés aléatoirement. Or cette initialisation peut modifier les résultats. Malheureusement, le temps me fera défaut pour réaliser cela[3].

[3]Une telle campagne de mesures demanderait plus de trois semaines de calculs à l'ordinateur pour moyenné sur seulement 5 courbes.

Partie 4

Physics Informed Neural Network (PINN)

Dans cette partie, nous allons nous intéresser à une autre architecture de réseau de neurones. L'objectif est de voir si, avec une architecture plus simple, il est possible d'obtenir des résultats corrects. Pour cela, nous expliquerons le principe de fonctionnement ainsi que son implémentation. Nous verrons ensuite les limites de cette méthode.

4.1 Cadre conceptuel

Revenons un peu sur le fonctionnement d'un réseau de neurones décrit au paragraphe 3.1. Lors du parcours "classique" du réseau de neurones, toutes les opérations effectuées sont dérivables (à condition que les fonctions d'activations le soit). Si on est capable de suivre toutes ces opérations, on peut partir des valeurs de sortie et remonter tout le réseau pour obtenir une dérivée "analytique" d'une sortie par rapport à une valeur d'entrée.

Par analytique, j'entends sans calculer deux valeurs d'une fonction f autour d'une entrée x, et ensuite approché la dérivée par $\frac{df}{dx} \approx \frac{f(x+\delta x/2)-f(x-\delta x/2)}{\delta x}$.

On est donc capable d'obtenir les dérivées partielles sortant d'un réseau de neurones en s'affranchissant de certaines erreurs numériques habituelles. Des personnes ont alors eu l'idée ([16], [17] ...) de rentrer dans la fonction de coût les équations des milieux continus et les conditions aux limites. Lors de l'entraînement, le réseau de neurones va alors chercher à se rapprocher d'une solution du problème physique considéré.

L'objectif de cette fin de stage est de voir l'efficacité de cette architecture dans le cadre d'un effondrement granulaire simple en 2D. Pour cela, nous allons **redéfinir le problème** pour pouvoir l'adapter à l'architecture. En effet, on n'utilise plus de particules, il va donc falloir introduire une équation de conservation de la masse. Pour cela on considère l'aspect biphasique du probléme. Les grains sont mélangés avec de l'air. On introduit ϕ, la proportion de sable dans un volume infinitésimal. On peut montrer en utilisant le théorème de transport de Reynold et la **conservation de la quantité de matière** que ϕ doit vérifier l'équation suivante :

$$\frac{\partial \phi}{\partial t} + \vec{\nabla} \cdot (\phi \vec{u}) = 0 \tag{4.1}$$

Où $\vec{u}$ est la vitesse du sable. L'équation de **conservation de la quantité de mouvement** prend quand à elle la forme :

$$\rho\phi(\frac{\partial u}{\partial t} + (\vec{u} \cdot \vec{\nabla})\vec{u}) = \rho\phi\vec{g} - \vec{\nabla}p + \vec{\nabla} \cdot \bar{\bar{\tau}} \tag{4.2}$$

Le cisaillement sera traité avec une **loi visqueuse** type $\mu(I)$ avec la régularisation de Frigaard et Nouar (voir [6]) c'est à dire $\bar{\bar{\tau}} = \eta\dot{\varepsilon}$ avec $\dot{\varepsilon} = \frac{1}{2}(\overline{\overline{\nabla u}} + \overline{\overline{\nabla u}}^T)$ et la viscosité est donnée par :

$$\eta = \frac{\mu_s p}{||\dot{\varepsilon}|| + \lambda} + \frac{\Delta\mu p}{I_0\sqrt{\phi p} + ||\dot{\varepsilon}|| + \lambda} \tag{4.3}$$

Il serait ici beaucoup plus compliqué de prendre une loi élastoplastique comme précédemment car il ne sera pas possible d'utiliser un shéma essais-correction comme dans la MPM. La loi visqueuse reprend les grandes lignes de la loi précédente en mettant un seuil de contrainte. Ce seuil est "adouci" par la régularisation afin d'avoir des solutions stables. Les valeurs numériques des différentes constantes ont été prises dans l'article de Chauchat [12].

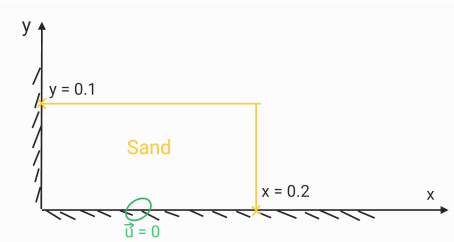

Figure 4.1: Conditions initial utilisé pour le PINN

Les **conditions aux limites** ont été fixées pour que la vitesse soit nulle au bord (no slip), que la pression en l'absence de sable soit la pression atmosphérique et que le tas de sable à l'instant initial soit de hauteur $h = 0.1m$, longueur $L = 0.2m$ avec une valeur de phi de 0.6. Ce tas de sable est adossé à un mur placé en $x = 0$ (voir figure 4.1)

4.2 La mise en application

Nous venons de voir ce qu'est un PINN et avons mis en forme le problème pour qu'il s'écrive toujours sous forme d'égalité. Je vais maintenant implémenter ces idées en python. Pour cela, j'utilise utiliser la bibliothèque pytorch. Cela permettra d'avoir un réseau de neurones qui soit bien optimisé et utilisant des modules codés en C++ donc plus rapides. Le code[1] est disponible en annexe A et sera régulièrement cité. Ce code définis une classe permettant de créer des objets PINN.

On définit d'abord **l'instanciation de la classe**. On fournit à la classe un "dataloader" c'est à dire un objet avec les données de positions et de temps que l'on veut tester lors de l'entraînement et qui regroupe ces données par batch. Un batch est une partie des données choisies de manière aléatoire. On essaie ensuite de voir si un GPU est disponible (ligne 4). On définit également la fonction de coût (ls ligne 20), le réseau de neurones et les optimiseurs. On reviendra sur ces derniers un peu plus tard. Le réseau a une architecture assez simple, 3 entrées (x,y,t), 15 couches cachées et 4 sorties (u, v, ψ et p). Les fonctions d'activation sont des tangentes hyperboliques.

Une méthode "function" (lignes 53 à 107) est ensuite définie. Elle permet non seulement de faire le parcours du réseau mais vient également traiter les données de sorties afin d'avoir les valeurs des équations du mouvement avec tous les termes à gauche.

Ces dernières valeurs permettent, en les comparant à 0, de calculer les fonctions de coût associées aux équations de la physique à l'aide de la méthode "mean square error" disponible avec torch (lignes 132 à 134). Dans **la méthode loss_function**, on vient également comparer les valeurs prédites par le réseau aux valeurs théoriques au bord et à l'instant initial (lignes 120 à 130).

On peut alors générer les boucles d'entraînement qui seront appelées dans **la méthode train**. La première boucle est celle utilisant **l'optimiseur Adam** [13]. Celui ci à l'avantage de converger relativement rapidement vers le minimum de la fonction de coût de manière assez stable. En effet, utiliser directement l'optimiseur Limited Memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) [3] a tendance à faire diverger

[1] Le code est inspiré de la vidéo de Computationnal Domain sur les PINN.

certains éléments des matrices du réseau. Cependant, ce deuxième optimiseur est plus précis et sera utilisé dans un deuxième temps lors de l'entraînement (ligne 191)

Une dernière méthode permet de sauvegarder toutes les données de la classe dans des fichier json et pt qui peuvent ensuite être lus lors de l'instanciation.

4.3 Avantages et limitations

Le principal avantage de cette algorithme est sa simplicité. Le fait qu'il n'est pas nécessaire de se baser sur des simulations permet d'accélérer sa mise en place en enlevant les problèmes de compatibilité rencontrés par exemple avec le GNS. La simplicité du réseau permet aussi de le mettre en place et de l'adapter très facilement. En revanche, cette simplicité vient avec des problèmes.

Une **première limite** vient de l'architecture même de l'algorithme. Or le réseau apprend la physique de l'écoulement dans son intégralité, il ne s'intéresse pas juste au passage d'un instant t à $t + dt$. Cela permet de limiter la taille de l'entrée et donc du réseau mais pose aussi problème. Le réseau apprend toute l'histoire de l'écoulement et la situation à point donnée (de l'espace et du temps) n'est pas locale, elle dépend aussi de toutes les conditions aux limites.

Cela veut dire qu'adapter l'algorithme, afin qu'il prenne en compte toutes les conditions aux limites imaginables, n'est pas possible. L'algorithme devra être réentrainer sur chaque situation. Cela rend le développement pour des situations réelles trop lourd en calcul. Ce serait plutôt un algorithme utilisable pour des situations de laboratoire où l'on ne fait varier qu'un nombre restreint de paramètres.

De plus, le réseau n'arrive pas à apprendre la physique correctement comme le montre la figure 4.2. Il semble donc que cette architecture soit un échec complet.

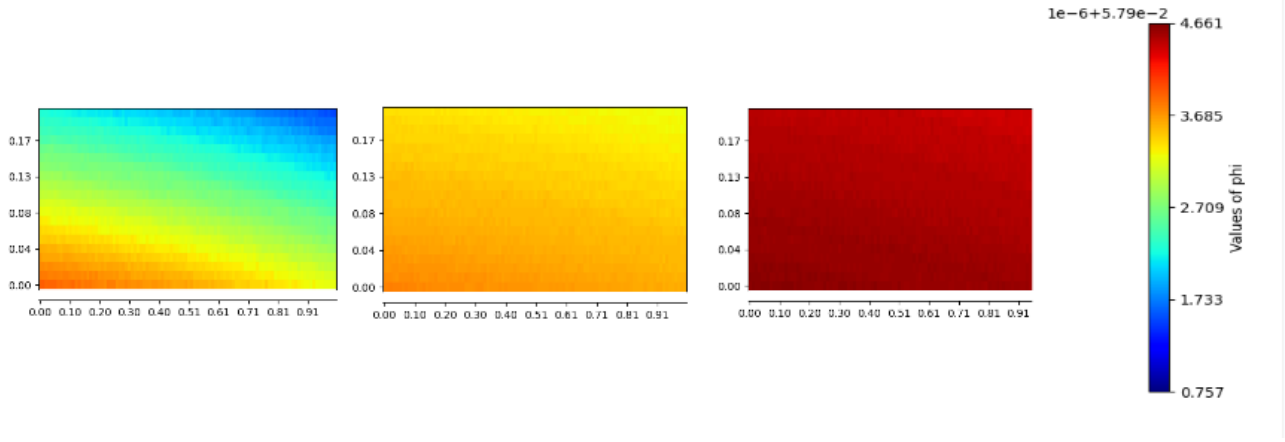

Figure 4.2: Rollout PINN après 1 semaine d'entraînement

Il est cependant possible de lui trouver des situations où elle pourrait être mieux adaptée. On a par exemple réfléchis à une situation un peu différente : le charriage lors de crues. Il serait alors possible d'entraîner dans un premier temps le réseau sur la situation stationnaire, les données étant disponible en allant faire des mesures dans la rivière. Avec des informations de terrains le réseau devrait réussir à apprendre l'écoulement comme le montre [18]. A partir de ce réseau, l'idée serait de relancer l'entraînement avec des conditions aux limites typique de crues et les équations régissant la physique. Le réseau devrait alors évoluer de manière à simuler cette nouvelle situation exceptionnel. Pour tester la validité de cette méthode, il serait possible de comparer les résultats à des expériences faites dans un canal artificiel comme celui de l'INRAE à Grenoble. Si cette méthode fonctionne, elle permettrait d'obtenir des simulations sur des portions de rivières plus longues que celles simuler avec des méthodes traditionnelles

Partie 5

Conclusion

5.1 Volet scientifique

L'objectif de mon stage était d'explorer la capacité de différents réseaux neuronaux à reproduire des expériences simples d'avalanches granulaires. Cet objectif doit être vu comme une preuve de concept, c'est à dire la première étape d'un projet qui cherche à émuler des avalanches (et autres écoulements gravitaires) en s'affranchissant du temps de calcul des simulateurs classiques et de permettre des prévisions en temps réel. Dans ce cadre là, une première conclusion est que les GNS ne sont peut-être pas la meilleure option. Leurs complexités les rendent puissants mais trop lourds à utiliser ou à modifier pour prendre en compte des géométries spécifiques.

Un deuxième point concerne les méthodes d'apprentissages. Ils semblent que, forcer l'apprentissage de certaines simulations moins bien comprises par le réseau, soit bénéfique non seulement pour éviter des écart de précisions entre différentes simulations, mais aussi pour que l'apprentissage global soit meilleur au final. Pour confirmer cela, il serait nécessaire d'avoir plus de données. Il serait aussi intéressant d'intégrer cet aspect directement dans le code original et pas à l'aide d'un code secondaire lançant les différentes simulations et entraînements via des "subprocess". L'algorithme perd en efficacité car il y a plus d'étapes et celles ci sont codées en python ce qui est moins efficace.

Enfin, les PINN sont des modèles intéressant mais trop contraint après l'entraînement pour pouvoir les utiliser pour simuler des écoulements soudains. En effet, donner seulement la physique dans la fonction de coût n'est pas suffisant pour apprendre un écoulement à partir d'une situation aléatoire. En revanche, c'est peut-être une piste pour des phénomènes de charriage. Il serait alors possible de partir d'un réseau connaissant déjà un écoulement, le cas stationnaire avant la crue, pour estimer le cas lors d'une crue. On pourrait alors obtenir des simulations sur des portions de rivière plus importantes qu'en utilisant des méthodes classiques.

5.2 Volet personnel

D'un point de vue personnel, ce stage fut très enrichissant. Le fait de partir à l'étranger dans une région où je ne parle pas la langue locale m'a permis de me rendre compte de la barrière linguistique. Les échanges ont toujours été possibles en anglais mais il a été plus difficile de s'intégrer socialement. Cela me pousse à reconsidérer certaines destinations que j'avais en tête pour de prochains stages.

J'ai découvert les subtilités de la recherche en particulier sur les problèmes de simulations. Par exemple, certaine choses qui paraissent simples peuvent parfois prendre beaucoup de temps à cause de problèmes de compatibilité de fichiers. Il faut être patient et méthodique pour résoudre ces difficultés que j'ai surtout rencontrer lors des deux premières semaines de stage. Cet aspect fut heureusement suivi de plus de temps à coder et à résoudre de "vrais problèmes". J'ai été intéressé à résoudre ces problèmes informatiques l'étude des problèmes théoriques. Entre autre, j'ai particulièrement apprécié les échanges que j'ai eu avec Hugo

Rousseau sur des modèles analytiques certes plus simples mais demandant une meilleure compréhension de la physique. Cela conforte mon projet d'étude plus axé sur les aspects théoriques de la physique.

Annexe A

Code PINN's

L'intégralité de ce que j'ai codé est disponible sur GitHub, que ce soit le travail effectué sur le GNS ou celui liée au PINN. Le code auquel je me réfère est quand à lui disponible ci dessous :

```
class NavierStokes():
    def __init__(self, dataloader, fromJsonFile = None):

        self.device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
        self.dataloader = dataloader
        self.sizeBatch = dataloader.batch_size

        #null vector to test against f and g:
        self.null = torch.zeros((self.sizeBatch, 1)).to(self.device)
        # initialize network:
        self.network()
        if fromJsonFile is not None :
            with open(fromJsonFile, 'r') as file :
                data = json.load(file)
            self.net.load_state_dict(torch.load(data['net']))
            self.ls = data['loss']
            self.iter = data['iter']
            self.ls_fct_steps = data['lossFctSteps']
        else :
            self.ls = 0
            self.iter = 0
            self.ls_fct_steps = []
        self.net = self.net.to(self.device)

        self.LBFGS_optimizer = torch.optim.LBFGS(self.net.parameters(),
                                                 lr=0.0001,
                                                 max_iter=1000, max_eval=None,
                                                 history_size=50,
                                                 tolerance_grad=1e-08,
                                                 tolerance_change=0.01 * np.finfo(float).eps
    ,
                                                 line_search_fn="strong_wolfe")

        self.adam_optimizer = torch.optim.Adam(self.net.parameters(), lr=0.0005)
        self.mse = nn.MSELoss()

        self.ls_average = 0

    def network(self):

        self.net = nn.Sequential(
            nn.Linear(3, 20), nn.Tanh(),
            nn.Linear(20, 20), nn.Tanh(),
```

```
            nn.Linear(20, 20), nn.Tanh(),
            nn.Linear(20, 20), nn.Tanh(),
            nn.Linear(20, 20), nn.Tanh(),
            nn.Linear(20, 20), nn.Tanh(),
            nn.Linear(20, 20), nn.Tanh(),
            nn.Linear(20, 20), nn.Tanh(),
            nn.Linear(20, 20), nn.Tanh(),
            nn.Linear(20, 20), nn.Tanh(),
            nn.Linear(20, 3))

    def function(self, x, y, t):

        #Get the results of the NN
        res = self.net(torch.hstack((x, y, t)))
        psi, p, phi = res[:, 0:1], res[:, 1:2], res[:, 2:3]
        p *= 1e5
        if isnan(psi[0].item()) :
            print('psi')
        elif isnan(p[0].item()) :
            print('p')
        elif isnan(phi[0].item()) :
            print('phi')

        #Modify and get related results
        u = torch.autograd.grad(psi, y, grad_outputs=torch.ones_like(psi), create_graph=True
    )[0] #retain_graph=True,
        v = -1.*torch.autograd.grad(psi, x, grad_outputs=torch.ones_like(psi), create_graph=
    True)[0]

        u_x = torch.autograd.grad(u, x, grad_outputs=torch.ones_like(u), create_graph=True)
    [0]
        u_xx = torch.autograd.grad(u_x, x, grad_outputs=torch.ones_like(u_x), create_graph=
    True)[0]
        u_xy = torch.autograd.grad(u_x, y, grad_outputs=torch.ones_like(u_x), create_graph=
    True)[0]

        u_y = torch.autograd.grad(u, y, grad_outputs=torch.ones_like(u), create_graph=True)
    [0]
        u_yy = torch.autograd.grad(u_y, y, grad_outputs=torch.ones_like(u_y), create_graph=
    True)[0]
        u_t = torch.autograd.grad(u, t, grad_outputs=torch.ones_like(u), create_graph=True)
    [0]

        v_x = torch.autograd.grad(v, x, grad_outputs=torch.ones_like(v), create_graph=True)
    [0]
        v_xx = torch.autograd.grad(v_x, x, grad_outputs=torch.ones_like(v_x), create_graph=
    True)[0]
        v_xy = torch.autograd.grad(v_x, y, grad_outputs=torch.ones_like(v_x), create_graph=
    True)[0]

        v_y = torch.autograd.grad(v, y, grad_outputs=torch.ones_like(v), create_graph=True)
    [0]
        v_yy = torch.autograd.grad(v_y, y, grad_outputs=torch.ones_like(v_y), create_graph=
    True)[0]
        v_t = torch.autograd.grad(v, t, grad_outputs=torch.ones_like(v), create_graph=True)
    [0]

        p_x = torch.autograd.grad(p, x, grad_outputs=torch.ones_like(p), create_graph=True)
    [0]
        p_y = torch.autograd.grad(p, y, grad_outputs=torch.ones_like(p), create_graph=True)
    [0]

        phi_x = torch.autograd.grad(phi, x, grad_outputs=torch.ones_like(phi), create_graph=
    True)[0]
```

```
        phi_y = torch.autograd.grad(phi, y, grad_outputs=torch.ones_like(phi), create_graph=True)[0]
        phi_t = torch.autograd.grad(phi, t, grad_outputs=torch.ones_like(phi), create_graph=True)[0]


        eps_dot = torch.sqrt_(u_x**2 + v_y**2 + 0.5 * (v_x + u_y)**2)
        e = (mu_s * p) / (eps_dot + lamb) + (delta_mu * p) / (I0 * torch.sqrt_(torch.abs_(phi * p)) + eps_dot + lamb)
        e_x = torch.autograd.grad(e, x, grad_outputs=torch.ones_like(e), create_graph=True)[0]
        e_y = torch.autograd.grad(e, y, grad_outputs=torch.ones_like(e), create_graph=True)[0]

        sigxx_x = e_x * u_x + e * u_xx
        sigyy_y = e_y * v_y + e * v_yy
        sigxy_y = 0.5 * e_y * (u_y + v_x) + 0.5 * e * (u_yy + v_xy)
        sigyx_x = 0.5 * e_x * (u_y + v_x) + 0.5 * e * (u_xy + v_xx)

        f = phi_t + u * phi_x + phi * u_x + v * phi_y + phi * v_y
        g = u_t + u * u_x + v * u_y + p_x - sigxx_x - sigxy_y
        h = v_t + u * v_x + v * v_y + p_y + rho * phi * G - sigyx_x - sigyy_y

        return u, v, p, phi, f, g, h

    def loss_function(self, x, y, t) :
        # u, v, p, g and f predictions:
        u_prediction, v_prediction, p_prediction, phi_prediction, f_prediction, g_prediction, h_prediction = self.function(x, y, t)

        # calculate losses
        u_loss = 0
        v_loss = 0
        phi_loss = 0
        p_loss = 0
        n, k = 0, 0
        for i in range(self.sizeBatch) :
            if x[i,0] < 0.0001 or x[i,0] > L - 0.0001 or y[i,0] < 0.0001 or y[i,0] > h - 0.0001 :
                u_loss += u_prediction[i,0]**2
                v_loss += v_prediction[i,0]**2
                n += 1
            if t[i,0] == 0 :
                k += 1
                phi_loss += (phi_prediction[i,0] - ci(x[i, 0], y[i,0]))**2
            if y[i,0] > 0.15 :
                p_loss = (p_prediction[-1] - p0)**2
        u_loss /= (n+1)
        phi_loss /= (k+1)
        f_loss = self.mse(f_prediction, self.null)
        g_loss = self.mse(g_prediction, self.null)
        h_loss = self.mse(h_prediction, self.null)
        self.ls = u_loss + v_loss + 10 * p_loss + 100 * phi_loss + 0.001 * f_loss + 0.001 * g_loss + 0.001 * h_loss
        # derivative with respect to net s weights:
        self.ls.backward()

    def closure(self):
        # reset gradients to zero:
        self.LBFGS_optimizer.zero_grad()
        x, y, t = next(iter(self.dataloader))
        x = x.to(self.device)
        y = y.to(self.device)
        t = t.to(self.device)
        self.loss_function(x, y, t)
```

```
147
148         self.iter += 1
149         self.ls_average += self.ls.item()
150         if self.iter % len(self.dataloader) == 0 :
151             self.ls_average /= len(self.dataloader)
152             if self.ls_average < min(self.ls_fct_steps) :
153                 torch.save(self.net.state_dict(), 'autobackup_lbfgs.pt')
154             self.ls_fct_steps.append(self.ls_average)
155             print('LBFGS EPOCH: {:}, Loss: {:0.6f}'.format(self.iter//len(self.dataloader),
    self.ls_average))
156             self.ls_average = 0
157         torch.nn.utils.clip_grad_norm_(self.net.parameters(), max_norm=0.3)
158         return self.ls
159
160     def LBFGS_train(self):
161
162         # training loop
163         self.net.train()
164         self.LBFGS_optimizer.step(self.closure)
165
166     def Adam_train(self, nb_epoch) :
167         min_ls = 1e12
168         for i in range(nb_epoch) :
169             self.ls_average = 0
170             for x, y, t in self.dataloader :
171                 x = x.to(self.device)
172                 y = y.to(self.device)
173                 t = t.to(self.device)
174                 self.adam_optimizer.zero_grad()
175                 self.loss_function(x, y, t)
176                 self.adam_optimizer.step()
177                 self.iter += 1
178                 self.ls_average += self.ls.item()
179             self.ls_average /= len(self.dataloader)
180             if i > nb_epoch // 3  and self.ls_average < min_ls :
181                 torch.save(self.net.state_dict(), 'autobackup_adam.pt')
182                 min_ls = self.ls_average
183             self.ls_fct_steps.append(self.ls_average)
184             print(f'Adam EPOCH {self.iter//len(self.dataloader)}, Loss: {self.ls_average:.6f
    }')
185
186     def train(self, nb_epochs_adam, nb_epoch_lbfgs) :
187         self.Adam_train(nb_epochs_adam)
188         self.net.load_state_dict(torch.load('autobackup_adam.pt'))
189
190         for i in range(nb_epoch_lbfgs) :
191             self.LBFGS_train()
192
193     def savePINN(self, fileName = 'autobackup.json') :
194         torch.save(self.net.state_dict(), 'model' + str(self.iter) + '.pt')
195         dico = {}
196         dico['net'] = 'model' + str(self.iter) + '.pt'
197         dico['loss'] = self.ls.item()
198         dico['lossFctSteps'] = self.ls_fct_steps
199         dico['iter'] = self.iter
200         with open(fileName, 'w') as file :
201             json.dump(dico, file, indent=2)
```

Bibliographie

[1] Blatny. *Modeling the mechanics and rheology of porous and granular media: an elastoplastic continuum approach*. Lausanne, EPFL, 2023.

[2] Jiang Chenfanfu, Schroeder Craig, Teran Joseph, Stomakhin Alexey, and Selle Andrew. The material point method for simulating continuum materials. *Digital library*, 2016.

[3] Liu Dong C. and Nocedal Jorge. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 1989.

[4] D. C. Drucker and W. Prager. Soil mechanics and plastic analysis for limit design. *Quarterly of Applied Mathematics*, 1952.

[5] Sakir Amiroudine et Jean-Luc Battaglia. *Mécanique des fluides, 4ème édition*. Dunod, 2022.

[6] I.A. Frigaard and C. Nouar. On the usage of viscosity regularisation methods for visco-plastic fluid flow computation. *J. Non-Newton. Fluid Mech*, 2005.

[7] Rousseau Gauthier, Métivet Thibaut, Rousseau Hugo, Daviet Gilles, and Bertails-Descoubes Florence. Revisiting the role of friction coefficients in granular collapses: confrontation of 3-d non-smooth simulations with experiments. *J. Fluid Mechanics*, 2023.

[8] Rousseau Hugo, Gaume Johan, Blatny Lars, and P. Lüthi Martin. Transition between mechanical and geometric controls in glacier crevassing processes. *Geophysical research letters*, 2024.

[9] Gaume J., Gast T., Teran J., van Herwijnen A., and Jiang C. Dynamic anticrack propagation in snow. *Nature Communications*, 2018.

[10] Gaume Johan and Ruegg Peter. Quand les pierres commencent à rouler. *Swiss Science Today*, 2024.

[11] Wolper Joshuah, Gao Ming, P. Lüthi Martin, Heller Valentin, Vieli Andreas, Jiang Chenfanfu, and Gaume Johan. A glacier–ocean interaction model for tsunami genesis due to iceberg calving. *Communications Earth and Environment*, 2021.

[12] Chauchat Julien and Médale Marc. A three-dimensional numerical model for dense granular flows based on the $\mu(i)$ rheology. *Journal of Computational Physics*, 2014.

[13] Erdmann Martin, Glombitza Jonas, Kasieczka Gregor, and Klemradt Uwe. *Deep Learning for Physics Research*. World Scientific, 2021.

[14] REBETEZ Martine, KUGON Ralph, and Pierre-Alain BAERISWYL. Climate change and debris flows in high mountains regions. *Climatic Change*, 1997.

[15] O. Pironneau. Finite element method for fluids. *Wiley*, 1988.

[16] Sharma Push, Chung Wai Tong, Akoush Bassem, and Ihme Matthias. A review of physics-informed machine learning in fluid mechanics. *Energies*, 2023.

[17] Cai Shengze, Mao Zhiping, Wang Zhicheng, and Yin Minglang. Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica*, 2021.

[18] Nakamura Taichi and Fukagata Koji. Robust training approach of neural networks for fluid flow state estimations. *Heat Fluid Flow*, 2022.

[19] Krishna Kumar YongjinChoi. Graph neural network-based surrogate model for granular flows. *Journal of Computational Physics*, 2024.

[20] Li Yunzhu, Wu Jiajun, Tedrake Russ, Tenenbaum Joshua B., and Torralba Antonio. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *ICRL*, 2019.