Rapport de stage

Simulations multi-échelle d'électromagnétisme sur des domaines complexes

Faculty of Mathematics
University of Strasbourg

Marie Sengler
Tutors : Emmanuel Franck and Victor Michel Dansac

Date : 19/08/2024

Contents

1 Introduction

Lors de ce semestre, j'ai pu travailler sur la thématique des PINNS pour Physics Informed Neural Network. L'objectif était de reproduire et d'améliorer des résultats de résolution d'équation aux dérivées partielles provenant de problème classique d'électromagnétisme utilisant la méthode PINNs. Les problèmes étaient présentés dans un rapport d'une étudiante ayant fait son stage de fin d'études dans l'entreprise Thales [2] : elle y présente les différents problèmes, la méthode des PINNS, les différentes mesures qu'elle obtient. Les mesures utilisées sont la MSE, MAE, l'erreur maximum, et le temps.

Lors du projet, on avait implémenté trois problèmes d'électromagnétisme à l'aide de scimba utilisant la méthode des PINNs. On avait pu modifier les différents paramètres du réseau, les différents poids des différentes fonctions de perte dans le but de trouver les optimales et avoir une erreur la plus petite possible entre la solution exacte et celle calculée avec les PINNs. Nous présenterons à nouveau en détails les différents problèmes dans une partie suivante. Le premier problème d'électrostatique a donné des bons résultats, le deuxième et troisième problèmes d'électromagnétisme avec deux matériaux différents étaient plus complexe à entraîner et ont donné quant à eux de moins bons résultats. En particulier le deuxième, dont les résultats correspondent à ceux du rapport sans adaptation des poids. Si on prend en compte l'adaptation des poids, les résultats sont en dessous de ceux de Thales.

Voici le chemin menant à la branche gitlab que j'ai utilisé pendant le stage : https://gitlab.inria.fr/sciml/scimba/-/tree/develop_marie?ref_type=heads

1.1 Objectifs

Le premier objectif du stage est donc d'améliorer les résultats obtenus avec les PINNS. Nous travaillerons sur le problème d'électrostatique avec deux matériaux, puis sur celui de magnétostatique. Pour cela, on peut modifier les paramètres, dont le taux d'apprentissage, le rapport entre les poids pour la fonction de perte au bord et dans le domaine, les poids des différentes fonctions de perte, les nombres de points de collocation dans le domaine et sur le bord, le nombre de neurones par couche et le nombre de couches, le ratio entre les deux réseaux de neurones, la fonction d'activation.

Le second grand objectif est d'implémenter un algorithme multi-level qui est une succession de réseaux à la suite l'un de l'autre. Il faudra ensuite l'optimiser pour pouvoir avoir de meilleurs résultats que ce que l'on avait avant. On devra aussi adapter différentes parties du code de scimba pour gérer plusieurs réseaux (comme par exemple pour la back-propagation). On devra aussi adapter nos calculs d'erreur et nos graphiques. Nous ferons des recherches d'optimisation d'abord à la main, puis on essayera d'automatiser certaines parties, en utilisant des algorithmes présentés dans un article scientifique. Nous détaillerons tout cela dans la partie dédié.

1.2 Présentation du lieu de stage et motivation de l'étude PINNs

Mon stage se déroule en juillet-août 2024 à l'IRMA (Institut de Recherche Mathématique Avancée), qui est un Institut de recherche en mathématiques sous la tutelle de l'Institut National des Sciences Mathématiques et de leurs Interactions du CNRS (INSMI) et de l'université de Strasbourg (UFR de Mathématique et d'Informatique). L'IRMA compte environ 120 membres répartis en 7 équipes de recherche.

Mes tuteurs sont Victor Michel-Dansac et Emmanuel Franck. Ils sont dans l'équipe travaillant sur scimba qui est un framework python pour la résolution d'équation utilisant des méthodes de machine learning comme les PINNs. Ils travaillent en collaboration avec l'entreprise Thales qui leur donne des projets. Thales est un groupe d'électronique français spécialisé dans l'aérospatial, la défense, la sécurité et le transport terrestre. L'entreprise a une importance mondiale et se consacre à la haute technologie. Ils ont besoin de fabriquer des pièces avec une haute précision et ont besoin d'étudier les phénomènes d'électromagnétisme. Pour cela, ils investissent dans la recherche en simulation numérique notamment avec la méthodes des éléments finis et plus récemment sur la récente méthode des PINNs.

Un des objectifs est d'avoir des erreurs les plus petites possible attenant presque la précision machine et qui ne soit pas trop coûteux en termes de temps et ressources utilisés. Nous avons pu échanger avec Julie Troyen et Flore Molenda de Thales par mail pour donner les premiers résultats trouvés avec l'algorithme greedy pour les PINNs. Elles ont pu aussi envoyer leurs solutions des problèmes calculés par éléments de façon plus précise. J'ai pu aussi assister à une réunion en visio-conférence, et une présentielle où mes tuteurs et eux de Thales ont pu discuter des avancées des deux côtés et de ceux qui peut-être fait par la suite. J'ai pu présenter aussi quelques-uns de mes résultats lors de la réunion.

Pendant le stage, je travaillais dans la salle informatique avec d'autres stagiaires de l'IRMA, et fais quelques rapports par slack ou en passant dans le bureau de mes tuteurs pour parler de mes avancées et de la suite du stage.

2 Retour sur les problèmes de Electrostatique et Magnétostatique

2.1 Rappel sur les PINNs

Les PINNs sont une méthode relativement récente introduite vers 2019, notamment par Raissi et al. dans *Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems.*[4] Les PINNS sont une technique de modélisation par IA qui résout des problèmes de physiques et d'ingénieurie utilisant des réseaux de neurones impliquant des EDP.

Une EDP peut être écrite sous la forme de

$$F(u(x)) = f(x), x \in \Omega$$

$$B(u(x)) = g(x), x \in \delta\Omega$$

avec Ω le domaine.

Pour approximer l'EDP, on utilise pour la fonction de perte du réseau de neurones l'addition pondérée de

- Une fonction de perte pour le résidu
- Une fonction de perte pour le résidu de bord pour les conditions de bord
- Une fonction de perte pour les conditions initiales

Nos cas d'études sont indépendants du temps, nous n'avons pas de conditions initiales.

Pour évaluer notre solution, nous utilisons des points de collocations choisi uniformément dans notre domaine et sur le bord de notre domaine pour éviter des irrégularité.

Soit N_F et N_B le nombre de points de collocations. Nous notons $\hat{u}_\theta(x)$ la solution approximative. Notre fonction de perte est donnée par:

$$L(\theta) = L_F(\theta) + L_B(\theta)$$

$$L_F(\theta) = \frac{1}{N_F} \sum_{i=1}^{N_F} ||F(\hat{u}_\theta(x_i)) - f(x_i)||^2$$

$$L_B(\theta) = \frac{1}{N_B} \sum_{i=1}^{N_B} ||B(\hat{u}_\theta(x_i)) - g(x_i)||^2$$

On peut ajouter des sous-domaines avec leurs propres propriétés physiques, en reliant les deux par des conditions aux interfaces et en ajoutant une fonction de perte pour cette interface.

2.2 Problème de Electrostatique avec deux matériaux

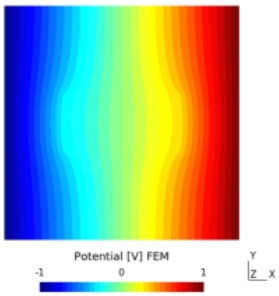

Figure 1: Solution du problème d'électrostatique avec deux matériaux calculé par éléments finis de Thales

On rappelle brièvement notre problème d'électrostatique avec deux matériaux. Dans notre premier modèle, la densité de courant électrique est prise nulle dans le vide. Notre domaine est $\Omega = [-1,1] * [-1,1]$ délimité par le bord Γ. Le champ électrique E suit les équations de Maxwell-Gauss et de Maxwell-Faraday. On a donc que le potentiel électrique suit une équation de Laplace généralisé $div(\epsilon grad(\phi)) = 0$. On prend $\epsilon = 1$, et des conditions de Dirichlet égal à -1 sur le bord gauche et à 1 sur le bord droit, et de Neumann homogène en haut et en bas. Ensuite, nous avons ajouté un dielectrique isolant dans le vide. Le sous-domaine est représenté par $\Omega_d = [-0.5, 0.5] * [-0.25, 0.25]$. On utilise la notation ϕ_v et ϕ_d pour le potentiel électrique respectivement dans le vide et le diélectrique, Γ pour l'interface, et $\epsilon_v = 1$, $\epsilon_d \neq 1$ pour les perméabilités relative du vide et du diélectrique.

Le système d'équation est donné par:

$$\begin{cases} \Delta\phi_v = 0, & \phi \in \Omega_v \\ \Delta\phi_d = 0, & \phi \in \Omega_d \\ \phi_v = \phi_d, & \phi \in \Gamma \\ \frac{\delta\phi_v}{\delta x} = \epsilon_r \frac{\delta\phi_d}{\delta x}, & \phi \in \Gamma_{left} \bigcup \Gamma_{right} \\ \frac{\delta\phi_v}{\delta y} = \epsilon_r \frac{\delta\phi_d}{\delta y}, & \phi \in \Gamma_{top} \bigcup \Gamma_{bottom} \end{cases}$$

Nous avons aussi un problème d'électrostatique similaire de base où on n'a pas de sous-domaine, où la solution évolue linéairement du bord gauche au bord droit.

2.3 Problème de Magnétostatique

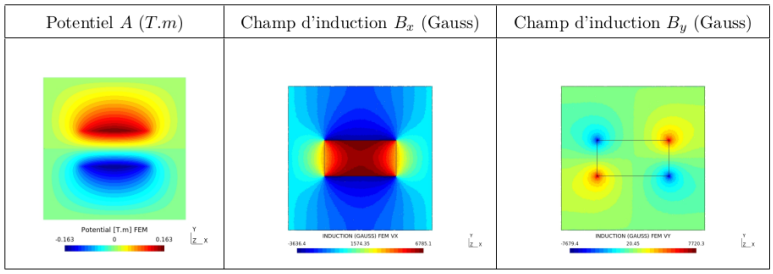

Figure 2: Solution du problème de Magnétostatique avec deux matériaux calculé par éléments finis de Thales

On rappelle le dernier problème présenté dans l'article [2]. On considère un problème de magnétostatique avec un courant de densité électrique égal à zéro. Notre domaine est le même que le problème précédent. On place un aimant dans un milieu qu'on considéra proche du vide. Le champ d'induction magnétique B suit une loi de Maxwell-Thomson et le chanp d'intensité magnétique H suit une loi de Maxwell-Ampère. L'induction B dérive du vecteur potentiel magnétique A.
Après plusieurs calculs vectoriels et de propriétés physiques, on se retrouvait avec le système suivant :

$$\begin{cases} \Delta A_m = & 0 & \in \Omega_m \\ \Delta A_v = & 0 & \in \Omega_v \\ \frac{\delta A_v}{\delta y} = & \frac{\delta A_m}{\delta y} & \in \Gamma_{left} \bigcup \Gamma_{right} \\ \frac{\delta A_v}{\delta x} = & \frac{1}{\mu_r} \frac{\delta A_m}{\delta x} & \in \Gamma_{left} \bigcup \Gamma_{right} \\ \frac{\delta A_v}{\delta x} = & \frac{\delta A_m}{\delta x} & \in \Gamma_{bottom} \bigcup \Gamma_{top} \\ \frac{\delta A_v}{\delta y} = & \frac{1}{\mu_r}(\frac{\delta A_m}{\delta y} - B_c) & \in \Gamma_{bottom} \bigcup \Gamma_{top} \\ A_m = & A_v & \in \Gamma \\ A_v = & 0 & \in \delta\Omega \end{cases}$$

2.4 Résultats

Nous allons préciser pour commencer les caractéristiques de la machine sur laquelle nous travaillerons :

```
Using device:  cuda
cuda devices:  1
cuda current device:  0
cuda device name:  Tesla V100-PCIE-16GB
```

Comme on a une discontinuité à l'interface, on utilise deux réseaux de neurones: l'un qui calcul la solution à l'intérieur du sous-domaine, l'autre à l'extérieur. Nous avons déjà fait les mêmes raisonnements lors du projet, mais

on va essayer de les approfondir un peu plus pour espérer avoir de meilleurs résultats.

Premièrement, on adapte quelque peu notre algorithme pour qu'il fonctionne correctement sur GPU avec pytorch. Notamment sur la V100, nous n'avons pas de sortie graphique, pour pouvoir visualiser les graphiques de prédiction et les enregistrer, on a besoin de sauvegarder les images avec la méthode savefig de matplotlib.

Dans nos premières mesures pour pouvoir optimiser notre problème, nous allons essayer différents taux d'apprentissage. On avait à la base un taux d'apprentissage de 10^{-2}. On le diminue progressivement. On choisit finalement un taux de $8 * 10^{-3}$. Ensuite, on va essayer de trouver un bon ratio entre nos deux réseaux de neurones. On fait pour cela plusieurs mesures et on observe que le deuxième réseau de neurones a besoin de plus de poids par rapport au premier pour obtenir de meilleurs résultats (1:6 voire plus). On fait varier ensuite les résidus correspondant à nos Laplaciens à l'intérieur de notre sous-domaine et à l'extérieur de celui-ci. Si on augmente le poids de celui à l'intérieur les résultats tendant à être meilleur jusqu'à un certain point. Pour que la fonction de perte descende plus vite, on peut normaliser les poids des différentes fonctions de pertes (à la place [2;4;4], on aura [0.2;0.4;0.4]).

On récapitule nos mesures dans le tableau dataBimatElectro.csv. Une autre idée que de le faire à la main aurait été de mettre les différents poids en paramètre du réseau de neurones, donnant alors une approximation des différents poids à mettre à nos fonctions de pertes. Ajouter des paramètres à notre réseau de neurones va complexifier l'entraînement. Pour cela, on s'est basé sur l'article de Xiang [7] sur le balancement des poids de la fonction de perte. Sur scimba, on a déjà un ajustement automatique entre les deux poids de la fonction de pertes pour le bord et de la fonction de perte sur le domaine. Mais il n'a pas été possible de les rajouter en paramètres.

On a corrigé plusieurs petits bugs sur le calcul de l'erreur de la dérivée Bx, By du problème de magnétostatique. L'erreur de Bx reste néanmoins assez grande jusqu'à plus de 0.5 (les valeurs allant d'environ -0.3 à 0.7), malgré une MSE/MAE plutôt bonne (environ $6 * 10^{-4}$ et $7 * 10^{-3}$ dans les meilleurs cas). On peut expliquer cela par une discontinuité forte entre le diélectrique et le vide des valeurs de Bx. Sur les bords bas et hauts du dialéctrique les valeurs sont maximales alors que ailleurs elles sont plus petites. En conséquence, les points à l'interface avec une forte discontinuité accumulent une forte erreur. L'erreur de By est aussi assez grande, quoique meilleure que pour Bx : on est sur du 0.1 voire 0.09 dans les meilleurs cas. L'erreur y est concentrée dans les quatre coins de l'interface, où se situent les agglomérats. La MSE et la MAE sont plutôt bonnes (environ $2 * 10^{-4}$ et $4 * 10^{-3}$ dans les meilleurs cas).

Une autre difficulté dans nos mesures est le caractère non-déterministe. Pour les mêmes paramètres, nous n'obtenons pas exactement les mêmes mesures de grandeur, bien que dans le même ordre de grandeur. Pour bien faire les choses, il faudrait donc faire plusieurs mesures avec les mêmes paramètres pour se faire une idée de la moyenne et des écart-types. Néanmoins, un entraînement prend beaucoup de temps, c'est pourquoi on fera que plusieurs mesures dans quelques

cas qui semblent intéressants (par exemple lors de nouveau record).

Pour le problème d'électrostatique avec deux matériaux, on obtient avec adaptation des poids, dans les meilleurs cas une erreur maximum de $1.2 * 10^{-2}$, une MSE de $8.2 * 10^{-5}$ et une MAE d'environ $9.1 * 10^{-4}$. Alors que sans adaptation de poids, on avait une erreur maximale de 0.1, une MSE d'environ $4.5 * 10^{-3}$ et une MAE d'environ $5 * 10^{-2}$. On peut remarquer que quand on met tous les poids à un, et qu'on ne favorise pas un réseau de neurones par rapport à un autre que sur le plot de notre prédiction, notre sous-domaine est mis à zéro, et qu'en-dehors la solution est trop plate et pas assez arrondi. Ce qui nous laisse suggérer qu'il faut mettre un beaucoup plus grand poids sur le réseau de neurones gérant le sous-domaine par rapport à l'autre. On rappelle que les meilleurs scores trouvés pour le même temps par l'entreprise Thales était une erreur maximum de $6.7 * 10^{-2}$, une MSE de $3.5 * 10^{-4}$, une MAE de $6 * 10^{-3}$. Avec plus de temps d'entraînement (2.5 heures) et l'adaptation des poids, ils obtenaient une erreur maximum de $5.9 * 10^{-2}$, une MSE de $3.3 * 10^{-6}$, et une MAE de $4.4 * 10^{-4}$.

Pour le problème de Magnétostatique, on obtient avec adaptation des poids au mieux une erreur maximale de $5.6 * 10^{-3}$, une MSE de $4.6 * 10^{-5}$, une MAE de $7.1 * 10^{-4}$. Alors que sans adaptation de poids, on avait une erreur maximale d'environ 0.02, une MSE d'environ $3.3 * 10^{-4}$ et une MAE de $4.21 * 10^{-4}$. On rappelle que Thales avait obtenu une MAE de 10^{-2}, une MSE de 10^{-3} et une erreur maximum de $1.3 * 10^{-2}$. Et, pour Bx, By, ils obtiennent une MSE/MAE d'environ 10^{-2}.

Concernant, la reproductibilité des résultats, si on fait plusieurs entraînements avec les mêmes paramètres, on n'obtient pas exactement les mêmes résultats, ils sont néanmoins du même ordre de grandeur.

Si on a besoin d'avoir de nouveau les mesures et graphiques (par exemple, on a fait des modifications dans les graphiques, on a des mesures de références plus précises). On peut mettre une variable `new_training` à False, le réseau passe l'entraînement et va directement aux graphiques et aux mesures. Nos données dont la fonction de perte, prédiction, sont enregistrées dans un fichier en .pth, pour pouvoir avoir accès plus tard.

3 Scimba

Pour mieux comprendre fonctionne scimba et pouvoir modifier de façon plus efficace le code, nous avons fait une carte des dépendance tous les fichiers utiles dans scimba. Nous les avons aussi lus pour comprendre comment et où était implémenté chaque partie nécessaire au fonctionnement de notre entraînement, et pour pouvoir ensuite les modifier si nécessaire.

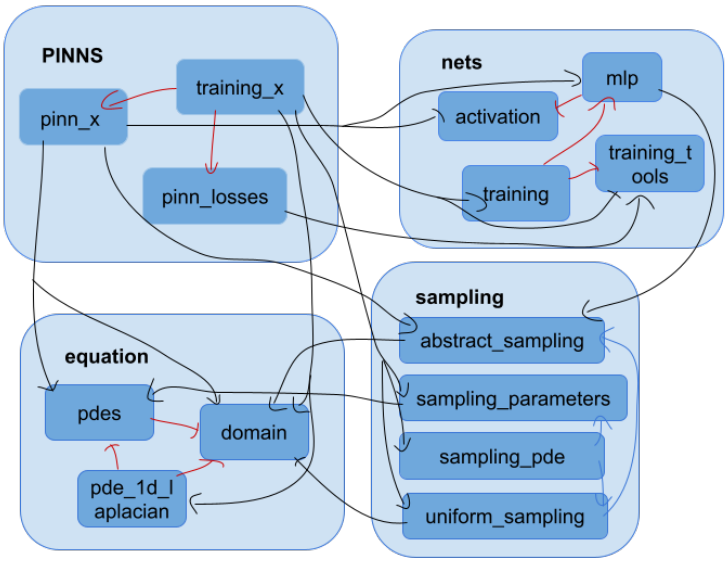

Figure 3: Diagramme représentant les différents liens entre les fichiers pour la partie PINNs de scimba

4 Algorithme greedy

Pour améliorer encore nos résultats, nous allons utiliser un réseau greedy qui est une liste de réseaux dans les premiers sont de simple MLP, et les derniers des Fourier Neural Network. Nous allons tout d'abord définir ce qu'est les réseaux greedy qui est une approche de résolution d'EDP récente.

4.1 Définition

D'après Wikipédia, l'algorithme greedy aussi connu sous le nom d'algorithme glouton en français, notion d'informatique théorique servant en optimisation, suit le principe de réaliser, étape par étape, un choix optimum local, afin d'obtenir un résultat optimum global. Par exemple, la descente de gradients, la recherche d'un arbre couvrant minimale peuvent être réalisée par un algorithme glouton.
D'après l'article intitulé "Greedy training algorithms for neural networks and applications to PDEs" du Journal of Computational Physics [5], l'algorithme greedy permettrai de résoudre les problèmes d'optimisation des réseaux de neurones lors d'un entraînement PINNS. L'idée est d'utiliser un algorithme glouton pour entraîner le réseau de neurones.
Soit une collection de vecteurs $\mathbb{D}$ dans un espace de Hilbert et une fonction cible u ou une fonction de perte $\mathcal{L}$, alors les algorithmes gloutons se rapprochent de u ou approximative minimise $\mathcal{L}$ par une combinaison linéaire finie d'éléments

de notre dictionnaire $\mathbb{D}$. Il existe comme type d'algorithme greedy les **relaxed greedy algorithm (RGA)** et **les orthogonal greedy algorithm (OGA)**. Dans scimba, ce sont pour l'instant les relaxed greedy algorithm qui ont été codé, dans le futur les orthogonal greedy algorithm le seront aussi.
On va présenter sans entrer dans les détails la base du relaxed greedy algorithm.
Cette algorithme se base sur les itérations : $\begin{cases} u_0 = 0, \\ g_n = argmax_{g\in\mathbb{D}}\langle g, \nabla\mathcal{L}u_{n-1}\rangle_H, \\ u_n = (1-\alpha_n)u_{n-1} - M\alpha_n g_n \end{cases}$
Le dictionnaire est supposé être symétrique. M est un paramètre de régularisation, et $\alpha_n = \min(1, \frac{2}{n})$.
L'architecture de base que l'on va utiliser pour nos problèmes est une série de réseaux de neurones. Les deux premiers sont des simples MLP, et les deux suivants sont des Fourier Neural Network (FNN), qui est un type de réseaux de neurones utilisant des transformations basées sur les séries de Fourier dans son architecture. Ils utilisent notamment des fonctions d'activation sinusoïdale, qui sont différentes des habituelles (relu, sigmoïde, tanh...). Avec une fonction d'activation en cosinus, on se trouve avec une sorte de décomposition de Fourier. Les FNN sont principalement utilisés dans le traitement des signaux et l'apprentissage des fonctions périodiques. Ils peuvent être combinés avec les PINNs et donnent les PIFNNs. Le choix des FNN s'impose par le fait que l'on essaye sur nos dernières étapes d'approximer des fonctions proche de zéro et oscillante. On pourra modifier cette architecture si besoin. Une des raisons de l'utilisation des FNN [3], et que les réseaux de neurones ont tendance à approximer d'abord les composantes à basses fréquentes de la fonction cible avant celle à hautes fréquences. Si les hautes fréquences sont importantes dans les composantes de la fonction, le réseau peut avoir du mal à converger pendant l'entraînement.
Pour résumer avec un shéma :

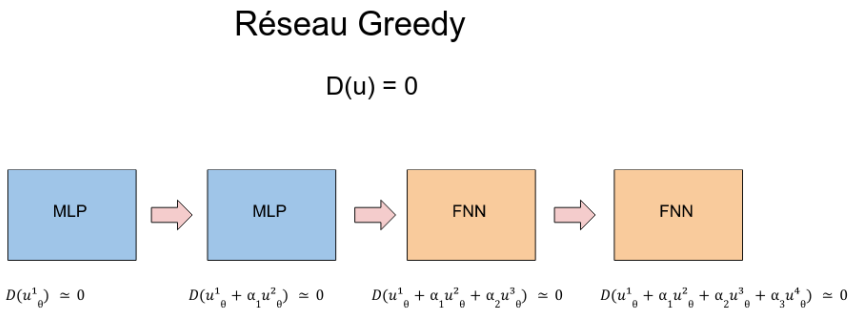

Figure 4: Schéma du fonctionnement d'un réseau greedy

Dans la pratique, on prendra la variable α de plus en plus petite à chaque étape. Par exemple, les paramètres α données dans un exemple de scimba sont : $\alpha_1 = 10^{-2}$, $\alpha_2 = 10^{-4}$ et $\alpha_3 = 10^{-5}$.

Nous avons à disposition deux types d'optimiseurs : Adam et un algorithme **LBFGS** pour (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) implémenté par torch. Les BFGS [1] sont des méthodes d'ordre supérieur à Adam et le coût pour une itération est plus important. On utilise alors le LBFGS qui permet d'économiser de la mémoire et d'optimiser ainsi l'entraînement. Il s'agit d'une méthode quasi-Newton qui pour minimiser la fonction objectif approxime la matrice Hessienne. Cette méthode a aussi la particularité de prendre en compte pour la descente de gradiants les gradients passés en plus de ceux actuels. Néanmoins, le LBFGS a tendance à converger vers des minima locaux dès les premières étapes d'entraînement, c'est pourquoi on le couple souvent avec l'optimiseur Adam. Dans nos entraînements, on a utilisé en général trois premiers quarts pour Adam, et le reste en LBFGS. Le LBFGS n'a pas montré de très grand gain dans la descente de la fonction de perte comme dans l'article [1], et une étape prend beaucoup plus de temps et de mémoire. On n'utilisera donc pas toujours le LBFGS dans nos entraînements.

4.2 Factorisation du code

Pour améliorer la lecture du code, éviter d'avoir des pages avec beaucoup de lignes et d'avoir du code similaire entre plusieurs fichiers, nous allons factoriser notre code. Nous créons un fichier `SpecificPlot.py`, où on va mettre toutes nos fonctions qui servent à plotter nos résultats. On ajoute des arguments à la fonction comme le nom, et si on utilise greedy ou pas, pour prendre en compte les spécificités des différents problèmes. En même temps, nous avons crée un ficher nommé `MyNetworksclass.py` où j'ai mis la classe `DoubleMLP` et `DoubleFourier`. Pour pouvoir utiliser le code dans ces fichiers, on les importe comme un module.

4.3 Implémentation sur le cas d'électrostatique

À partir d'un fichier modèle utilisant l'algorithme greedy, on modifie notre cas simple d'électrostatique pour l'adapter à la méthode greedy. Les deux problèmes étant compatibles, l'adaptation a pu se faire rapidement. Sur ce problème qui donnait déjà de très bons résultats, fonctionne bien avec l'algorithme greedy. Au bout de 60 secondes et seulement 5 epochs par réseaux, on obtient déjà une MSE de l'ordre de 10^{-5} et une erreur maximum et une MAE de 10^{-3}. Comme avant, on fait plusieurs mesures en modifiant à chaque fois divers paramètres.

Pour regarder comment notre solution évolue de réseaux de neurones en réseaux de neurones, après un entraînement de n epochs, on plotte notre solution, la carte du résidu, la carte de l'erreur.
On peut observer un contraste entre la carte du résidu qui s'approche de beaucoup de zéro et la carte de l'erreur qui a plutôt tendance à stagner. Dans notre cas, la solution a été calculée déjà assez précisément par notre premier réseau de neurones.

4.4 Implémentation sur le cas d'électrostatique avec deux matériaux et de magnétostatique

Ces deux problèmes s'avèrent plus délicats, on a en effet deux réseaux de neurones à cause de discontinuité à l'interface. Nous devons alors adapter notre suite de réseaux au fait qu'à chaque fois, on a deux réseaux en même temps (l'un pour notre sous-domaine et l'autre en dehors de celui-ci). On utilise alors notre classe `DoubleMLP_x`, et on va créer une classe `DoubleFourier_x` à partir de `Fourier_x` et en s'inspirant de la classe `DoubleMLP_x`.

Nos premiers essais ne sont pas très concluant, l'algorithme prend beaucoup de temps à s'éxcuter, est volumineux et ne donne pas de solution très précise. Notre fonction de perte a tendance à descendre plutôt bien. Sur notre cas d'électrostatique, on obtient des fonctions de pertes allant jusqu'à 10^{-12}. On observe qu'entre chaque réseau de neurones de notre série, la fonction de perte fait un saut.

4.4.1 Nouveaux graphiques et calcul d'erreur

Pour pouvoir visualiser la solution, le résidu et l'erreur à chaque étape pour le cas d'électrostatique et de magnétostatique avec deux matériaux, nous avons besoin d'adapter le plotter intitulé `plot_all_residuals_and_erreurs` qui prend en argument le trainer, le nombre de visualisation. Comme nous utilisons à chaque fois deux réseaux de neurones à cause de nos discontinuités, notre solution est séparée en deux parties. Nous réadaptons cette fonction, comme nous l'avons fait avec les autres plotters. Nous nous retrouvons comme dans le cas d'électrostatique avec un résidu qui continue à diminuer alors que l'erreur stagne à partir des réseaux Fourier. En conséquence, notre erreur reste beaucoup trop grande pour notre temps de calcul.

Pour mieux comprendre d'où venait les difficultés du réseau pour avoir de bons résultats, nous avons aussi calculé les erreurs intermédiaires entre chaque réseau de neurones.

Pour que ce soit pour le plot ou le calcul d'erreur, nous avons utilisées la méthode `setup_w_i_dict` de scimba qui prend en argument le SpaceTensor `x`, le paramètre mu, l'index du networks minimum et maximum. Cette méthode fait appel à la méthode `get_w_i`, prenant les mêmes arguments et qui rends la somme pondérée par alpha des différentes solutions de nos réseaux.

On a pu avoir la deuxième semaine de stage la solution calculée avec le logiciel d'éléments finis d'électrostatique de Thales plus précisément. Après analyse de l'erreur entre nos réseaux de neurones, que son comportement n'était pas très stable selon les paramètres : L'erreur ne diminuait pas toujours entre chaque réseau, elle avait tendance à stagner, voire à augmenter sur certaines mesures. Cela peut s'expliquer en partie que l'architecture du réseau de neurones qui fonctionnait bien sur d'autres problèmes, à l'air inadaptés sur le nôtre, qui apporte une complexité en plus avec des conditions aux interfaces discontinues et l'utilisation de deux réseaux de neurones. Il nous semble logique de chercher à optimiser l'architecture de nos réseaux de neurones.

4.4.2 Nombre d'epochs et de points de collocations

Nous avons fait encore une modification : Sur la technique greedy implémenté dans scimba, on pouvait changer le nombre de points de collocation de réseaux en réseaux de neurones, mais celui pour le nombre de collocation point sur le bord reste constant tout au long de l'entraînement. En s'inspirant de ce qu'il y a été fait pour le nombre de points de collocation, on modifie le fichier `greedy_training_x.py` pour pouvoir modifier le nombre de points de collocation sur le bord et pouvoir ainsi l'augmenter au cours de l'entraînement. Cela peut-être utile à l'entraînement, car l'erreur est principalement localisée sur le bord et les interfaces. Nous avons fait les mêmes manipulations pour pouvoir changer le nombre d'epochs par networks.

4.5 Recherche d'un bon réseau de neurone

Face aux difficultés d'apprentissage et que l'on a des nouvelles mesures nous permettant de mieux comprendre l'évolution de l'erreur et du résidu, nous allons tester de nouvelles structures de réseaux de neurones. Pour nos tests, on va modifier le nombre alpha, les structures de nos réseaux : on va essayer des structures différentes pour nos réseaux, changer le nombre de neurones par couche.

Les trois architectures différentes courantes pour les PINNs sont :

- Nombre constant de neurones par couche (Peut manquer de flexibilité pour capturer des relations complexes dans les données ou au contraire peut entraîner une sur-paramétrisation dans certaines couches, conduisant à un risque accru de sur-apprentissage.)
- Architecture progressive (petit -> grand -> petit)
- Architecture en sablier (grand -> petit -> grand)

Les deux dernières architectures offrent des avantages de flexibilité, mais sont plus compliquées à implémenter, et nécessitent plusieurs essaies pour trouver l'optimal. Une difficulté supplémentaire est que l'on utilise des réseaux déjà assez volumineux, et l'on doit faire attention aux dépassements de capacités de mémoire, comme ça a pu déjà arriver lors de nos entraînements. Ces dépassements de mémoire arrivent quand plusieurs personnes utilisent en même temps la V100. Comme les problèmes de mémoire prenaient trop de place à certains moments, on a fait quelques entraînements sur gaya. Gaya tourne sur CPU, mais a beaucoup plus de mémoire RAM que la V100. On aura donc des entraînements plus longs, mais on pourra utiliser des modèles potentiellement plus poussés. On indiquera par CPU ou GPU sur notre fiche de mesures pour savoir sur quelle machine a été fait l'entraînement.
Gaya offre la possibilité d'avoir des GPU, mais sur AMD. Pour avoir accès à cuda de pytorch pour utiliser pleinement le GPU, il aurait fallu installer un adaptateur tel que ROCm. En conséquence, quand on utilisera gaya se sera toujours sur CPU.

Pour gérer les problèmes de mémoires, on peut aussi réduire la taille de batch. Les capacités de parallélisation du CPU/GPU sont diminuées, l'entraînement est plus instable et le temps par epochs est plus long, mais cela permet de ne pas rester coincé dans des minima locaux et surtout l'entraînement consomme moins de mémoire. Le nombre d'epochs nécessaire pour avoir un bon résultat est aussi diminuer.

Les architectures progressives donnent d'assez bons résultats. Nous allons donc favoriser ce genre d'architecture pour les prochains entraînements.

Nous pouvons aussi jouer avec le type d'activation : Nous avons le choix entre tanh, sine, swish, et quelques autres. Dans nos tests, nous avons principalement utilisé sine et tanh. Nous avons aussi testé swish qui est une fonction d'activation de la famille ReLU et SiLU qui a donné d'assez bon résultat sur certains entraînements.

Pour les réseaux de Fourier, nous pouvons aussi modifier le nombre de features et l'écart-type de x. En général, d'un réseau FNN au suivant, j'augmente le nombre de features et je diminue l'écart-type.

4.6 Choix des alphas

On rappelle que pour la méthode greedy, on décompose notre solution en la somme pondérée de plusieurs réseaux de neurones.

$$\phi(x) \approx \sum_i \alpha_i \phi_i(x)$$

où ϕ_i sont les solutions approximatives données par les différents réseaux et α_i sont des poids. Pour déterminer les α_i, on peut procéder de plusieurs façons :

- En cherchant à la main
- Optimiser les alphas comme un paramètre pour minimiser le résidu en utilisant un algorithme de gradient.

Des alphas mal ajustés peuvent donner une importance disproportionnée à certains réseaux et permet d'équilibrer les différentes contributions. Dans un premier temps, nous chercherons les alphas à la mains, puis nous implémenterons des algorithmes de recherches des alphas.

4.7 Correction de la fonction de perte de bord

Nous avons pu observer en regardant les différents graphes de fonction de pertes que l'on pouvait avoir des sauts des fonctions de pertes entre les différents réseaux de neurones. Mais on a observé aussi que ces sauts provenaient principalement de la fonction de perte du bord, celle du résidu descend plus normalement. Après avoir fait ce saut vers le bas, la fonction de perte du bord pouvait stagner pendant le reste de l'entraînement. C'est pourquoi, j'ai pensé qu'il y pouvait avoir un problème d'implémentation concernant cette fonction

de perte. Le problème où la méthode greedy de base n'utilisé par de la fonction de perte de bord.

En analysant le code, nous avons remarqué que le résidu était multiplié par un loss factor et la loss de bord ne l'était pas. Je modifie donc la loss de bord pour être multiplié par le loss factor.

On essaye le code corrigé sur notre problème d'électrostatique simple, les différentes fonctions de pertes ne font plus de saut abrupte vers le bas. Nous essayons aussi sur le problème d'électrostatique avec deux matériaux, où la fonction de perte du bord avait tendance aussi à avoir ce genre de comportement. Les résultats sont beaucoup mieux. La fonction de perte descend correctement et la carte de l'erreur évolue de la même manière que le résidu, et descend mieux. Dans les réseaux suivant le premier les prédictions correspondent enfin aux endroits où le résidu était assez grand.

Les résultats étant un peu mieux nous pouvons passer à des choses plus compliquées pour améliorer l'entraînement.

4.8 Ajout de nouveaux graphiques

Jusqu'alors nous avons nos graphiques de bases avec la fonction de perte, la prédiction, la solution exacte, et l'erreur, ainsi que nos graphiques intermédiaires avec la prédiction, le résidu et l'erreur. Nous avons fait quelques modifications sur ces graphiques notamment au moment de la factorisation du code. Nous avons rajouté encore un ensemble de graphiques, nous prenons la prédiction et le résidu d'un réseau de neurones de manière isolée sans tenir compte de ce qui c'est passé avant. Ainsi, nous pouvons visualiser à quels endroits précisément, chaque réseau travaille et détecter des anomalies. Nous avons décidé d'abord d'utiliser des barres de couleurs différentes pour chaque sous-réseau, pour visualiser plus facilement si l'un des deux travaillait plus, mais cela apporter des difficultés de lecture supplémentaire. Nous avons alors décider de prendre le maximum et le minimum des deux pour avoir une barre de couleur unique au risque que l'on puisse moins voir les variations du réseau qui travailleraient moins.

Pour Bx et By, nous avons eu que l'erreur était principalement localisée en des points isolées sur le bord du sous-domaine, et pouvait être très grande comparée aux valeurs des autres points. C'est pourquoi on a voulu tester une normalisation des points en utilisant la racine carrée. On utilise pour cela `PowerNorm` de `matplotlib.color` en option de nos graphiques.

Pour les autres graphiques, on a aussi essayé avec ou sans double barre pour la couleur. On a alors moins vite au maximum/minimum des deux sous-réseaux, nous n'avons que les extrema globaux, mais l'échelle de couleur est plus cohérente.

Nous donnons un exemple de nos nouveaux graphiques :

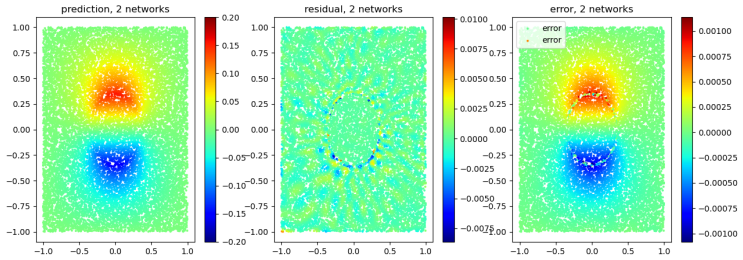

Figure 5: Prédiction au réseau n°2 du problème de Magnétostatique sur un cercle

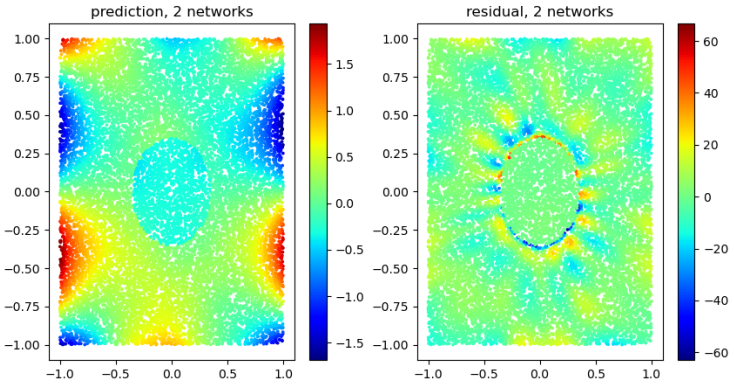

Figure 6: Prédiction isolée au réseau n°2 du problème de Magnétostatique sur un cercle

4.9 Correction graphiques/mesures de Bx/By

Pour Bx et By, j'ai essayé différents types d'interpolation (de base, j'étais en "cubic") de `scipy.interplate.griddata`. La méthode nearest a semblé donner de meilleurs résultats à part en quelques rares points, où l'erreur était encore plus grande qu'avant. On enlève ses valeurs anormales. Selon qu'on est sur le cas du rectangle ou du cercle, on obtient une division de l'erreur entre 2 et 10. Pour le cercle, l'erreur passe les 10^{-2} en erreur maximum. Pour le rectangle, les erreurs descendent plus difficilement : les erreurs dans les quatre coins du rectangle restent grandes, tournant au alentour de 0.1. L'erreur de By reste donc similaire, pour Bx, nous avons bien diminué l'erreur qui pouvait atteindre les 0.6.

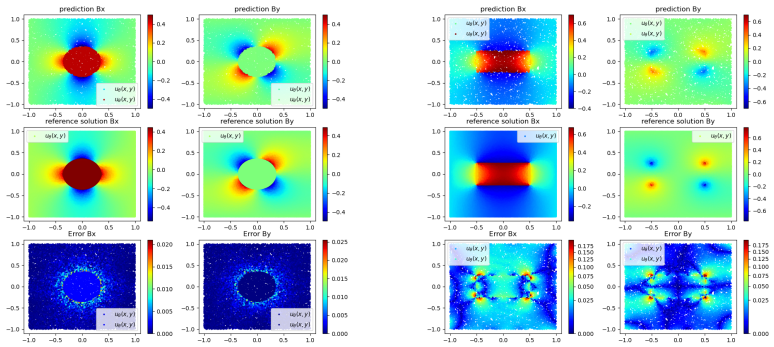

Figure 7: Correction des graphiques d'erreurs de Bx et By

4.10 Premier résultats

Comme ce qu'on avait fait avant, on lance plusieurs entraînements variants le nombre de points de collocations, l'architecture des réseaux, les alphas (certaines mesures sont faites avec la formule calculant les alphas décrits ci-dessous), ...

Pour le cas d'électrostatique, on arrive sur certaines mesures à des records en termes d'erreur. (une erreur maximum de 10^{-6} et une MAE de l'ordre de 10^{-7}, une MSE de l'ordre de 10^{-8} et une erreur minimum de l'ordre de 10^{-11}) Nous avons des résultats beaucoup plus précis qu'avant sur ce problème.

Pour le cas d'électrostatique avec deux matériaux, on obtient une erreur maximum légèrement meilleure de l'ordre de 10^{-2}, une MSE de l'ordre de 10^{-6}, une MAE de 10^{-4}, et une erreur minimum aussi meilleure de l'ordre de 10^{-8}.

Le problème de magnétostatique se trouve être plus délicat à entraîner. On a un peu plus de mal à l'entraîner. Les résultats correspondent un peu près à ceux que l'on avait sans greedy. Après plusieurs essaies rendant des résultats moyens, on a essayé la combinaison MLP, FNN et FNN qui a donnée de meilleurs résultats, surtout au niveau du deuxième FNN. Nous remarquons que la fonction de perte oscille beaucoup à partir de FNN. Nous obtenons une MAE de $3.2 * 10^{-4}$, une MSE de $3 * 10^{-6}$, une erreur minimum de $3 * 10^{-8}$, mais surtout une erreur maximum de seulement $2.6 * 10^{-3}$. Sur ce problème, le deuxième FNN a nettement amélioré l'erreur, mais notre fonction de perte qui est notre résidu ne descendait pas beaucoup. C'est aussi ce qu'à remarquer [1]. Nous avons fait en sorte que ce réseau capture spécifiquement les basses fréquences. Celui avant capturer les hautes, la fonction de perte descendait mieux, mais l'erreur a tendance à osciller, descendre plus doucement.

On remarque que l'erreur et le résidu à la fin de l'entraînement sur le problème d'électrostatique avec deux matériaux sont principalement concentré dans les quatre coins du bord. Dans les coins, la normal est mal définie, et ainsi que les conditions aux bords, ce qui rend plus difficile pour les réseaux d'apprendre

en ces points. Pour éviter une explosion dans ces points, on peut prendre l'arctangente de la fonction de perte.

4.11 Modification de schedulder

Sur notre problème de Magnétostatique, nous avons remarqué que la fonction de perte oscillait beaucoup surtout à partir des réseaux de Fourier. Les problèmes d'oscillation dans la fonction de perte sont souvent dus à un taux d'apprentissage inadapté. C'est pourquoi on utilise un schedulder, qui modifie la valeur du taux d'apprentissage au cours de l'entraînement. Dans scimba, le schedulder de base utilisé est le `StepLR` de torch, il diminue le taux d'apprentissage d'un certain taux à chaque nombre d'étapes. Nous avons essayé un autre schedulder le `ReduceLROnPlateau`. Ce schedulder détecte quand la metrics qu'on veut surveiller ne s'améliore plus (ici notre fonction de perte), et ajuste le taux d'apprentissage en conséquence. Il multiplie le taux d'apprentissage par un nombre entre 0.1 et 0.9 choisi par l'utilisateur qui peut aussi indiquer la patience, c'est-à-dire le nombre d'étapes d'apprentissage où notre metrics n'évolue pas avant de faire des modifications.
Nous commençons l'entraînement sur nos cas classiques avec un taux d'apprentissage un peu plus grand. On remarque qu'au premier réseau la fonction de perte peut descendre plus bas et plus vite. On fait plusieurs essais avec les deux schedulders pour trouver les paramètres optimaux. On a aussi repris le schedulder de base avec différents paramètres avec le nombre de step avant réduction et le decay le nombre de décalage pour notre taux d'apprentissage.

4.12 Calcul des alphas

Jusqu'à maintenant, nous avons essayé de trouver les alphas qui pondèrent la somme de chacun de nos réseaux de neurones à la main. Mais il existe des formules permettant de mieux approximer les alphas à un ordre de grandeur près. Ces formules sont présentées dans l'article [6] intitulé *Multi-stage neural networks: Function approximator of machine precision.*

Nous allons essayer d'adapter la formule générale à notre problème. Nous considérons tout d'abord un opérateur différentielle $\mathcal{N}$. Dans le cas de notre problème, nous avons que $\mathcal{N}(u) = \Delta u = \frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}$. En connaissant les fréquences dominantes fx_d et fy_d, la magnitude ϵ_1 de l'erreur est donnée par $\epsilon_1 = \frac{RMS(residu(x,u_0))}{(2\pi fx_d)(2\pi fy_d)RMS(\beta_m)}$ avec $\beta_m = \frac{\delta \mathcal{N}}{\delta u^{(m)}}|_{u=u_0}$

Pour déterminer β_m, nous allons calculer $\frac{\delta \mathcal{N}}{\delta u_{xx}}$ et $\frac{\delta \mathcal{N}}{\delta u_{yy}}$ avec $u_{xx} = \frac{\delta^2 u}{\delta x^2}$ et $u_{yy} = \frac{\delta^2 u}{\delta y^2}$.

$$\frac{\delta \mathcal{N}}{\delta u_{xx}} = \frac{\delta}{\delta u_{xx}}(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}) = 1$$
$$\frac{\delta \mathcal{N}}{\delta u_{yy}} = \frac{\delta}{\delta u_{yy}}(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}) = 1$$

Pour $k = 0$ et $k = 1$, $\frac{\delta \mathcal{N}}{\delta u_x} = \frac{\delta \mathcal{N}}{\delta u_y} = \frac{\delta \mathcal{N}}{\delta u} = 0$
Enfin, on a que $RMS(\beta_2) = 1$. Pour déterminer la fréquence dominante plus aisément, nous pouvons utiliser la relation avec notre magnitude ϵ du résidu qui après plusieurs étapes d'entraînement suit la loi $f_d \approx f_0 \epsilon^{-\frac{1}{6}}$, où f_0 est la fréquence dominante des données d'entraînement original.

La méthode pour déterminer la magnitude présentée dans l'article n'est pas ultra claire, et n'est pas adaptable à notre problème. La norme du terme source de la fonction est utilisée dans le calcul, or le nôtre est égal à 0. Or, à une étape de l'algorithme, nous avons besoin de diviser par cette norme.

Pour trouver la fréquence dominante de nos résultats, nous utilisons la fast fourier transform (fft) pour les images de numpy. Pour pouvoir faire cela, on a besoin d'interpoler nos points choisis uniformément pour en faire une grille utilisable pour la fft. On fait ensuite un décalage de notre fft pour centrer les basses fréquences et on prend la valeur absolue pour avoir la magnitude. Nous faisons fftfreq de numpy pour avoir les fréquences d'échantillonnage de la transformée de Fourier discrète. Nous faisons les mêmes opérations de décalages. On peut alors trouver les indices du maximum d'amplitude et de l'utiliser pour trouver la fréquence dominante selon x et y.

Néanmoins, on peut tomber sur des fréquences dominantes de 0. possible s'il n'y a pas de période ce qui pose des problèmes notamment au niveau de la division, c'est pourquoi on a choisi de prendre une valeur de 0.5, pour éviter que l'algorithme diverge.

On essayera aussi de déterminer les fréquences par essais et erreurs. On fait plusieurs mesures qu'on enregistre, puis on les compare entrer elles.
Nous savons que pour notre solution d'électrostatique, notre fréquence dominante est nulle en y, car on a aucune variation de la solution en y. Pour x, la solution évolue linéairement, on est alors plutôt dans des basses fréquences proches de zéro. Nous ne pouvons pas prendre f0x et f0y a zéro, car on divise par fx et fy pour obtenir nos alphas. C'est pourquoi on a commencé à chercher nos f0x et f0y optimaux inférieurs à 1. Pour le problème d'électrostatique des fréquences vers 0.1 - 0.3 ont l'air de donner de bons résultats.
Pour le problème d'électrostatique avec deux matériaux déterminés, la fréquence dominante est un peu plus compliquée. Pour cela, on va déterminés les fréquences spatiales fondamentales de notre domaine extérieur et intérieur. Notre domaine extérieur est un rectangle de $[-1, 1] * [-1, 1]$. On a donc une longueur d'onde fondamentale $\lambda_x = 2$ et une fréquence spatial fondamentale $k_x = \frac{\pi}{2}$. Pour notre domaine intérieur $[-0.5, 0.5] * [-0.25, 0.25]$, on aura une longueur d'onde fondamental de $\lambda_x = 1$, $\lambda_y = 0.5$, et donc d'une fréquence spatial fondamentale $k_x = \pi$, $k_y = 2\pi$. Les fréquences spatiales dominantes peuvent être approximées par les premiers harmoniques, on peut en déduire qu'ils sont de l'ordre de l'unité. Pour le problème de magnétostatique, on raisonne pareil.

Si la fonction source a une certaine fréquence dominante (par exemple, elle oscille avec une certaine fréquence), alors la solution $u(x)$ doit osciller avec la même fréquence dominante pour que l'équation différentielle soit équilibrée. C'est pourquoi travailler sur les fréquences a un intérêt pour avoir une solution

précise.

Néanmoins, nous avons plusieurs points divergents par rapport à l'article, qui peut expliquer pourquoi la méthode présentée fonctionne mal :

- Notre fonction source est nulle, donc la fréquence est nulle
- Nous avons des conditions aux bords et aux interfaces

Avoir des conditions aux bords et aux interfaces influence sur l'amplitude, la forme et la distribution fréquentielle, en conséquence cela peut compliquer la satisfaction de l'équation dans le domaine des fréquences.

Dans le but d'avoir une équation plus adoptée pour implémenter notre calcul de fréquence. Nous allons créer un nouveau fichier intitulé `frequence.py` avec un modèle plutôt simple : $\Delta u = -2\pi^2 \sin(\pi x) \sin(\pi y)$ avec des conditions de Dirichlet nulle. Notre solution exacte est donnée par $sin(\pi x) sin(\pi y)$ sur le pavé $[-1, 1] \times [-1, 1]$. Ce problème est plus proche de ceux que l'on a dans l'article. Les conditions aux bords sont nulles et la fonction source, n'est pas nulle et a pour fréquences dominantes : $fx = fy = \pi$.

4.13 Magnétostatique sur un cercle

Nous avons vu que même si l'erreur descendait bien, le réseau avait du mal à apprendre la solution dans les coins où la normal est mal définie. C'est pourquoi on reprend notre problème de Magnétostatique, mais à la place d'un aimant en forme de rectangle, on le remplace par un cercle. On adapte notre domaine pour y mettre un cercle : on change notre fonction de distance signée, on va utiliser l'équation d'un cercle classique : $(x-a)^2 + (y-b)^2 - R^2$. Pour notre problème, le centre est $(0, 0)$, le rayon est de 0.35.

La solution du problème a été calculée par Thales avec un logiciel d'éléments finis et ressemble à :

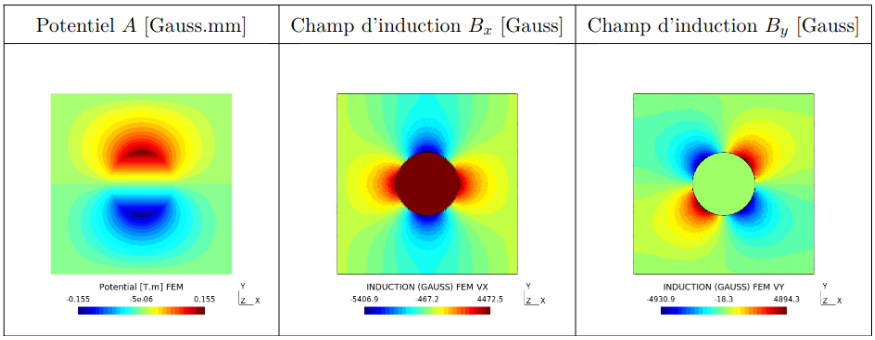

Figure 8: Solution du problème de Magnétostatique avec deux matériaux sur un cercle calculée par éléments finis de Thales

Pour notre domaine, on remplace notre rectangle, par un cercle entier ayant qu'un seul label, de rayon 0.35 et de centre $(0, 0)$. Pour définir la courbe

paramétrique de notre cercle, on utilise les fonctions $a+R\cos(\theta)$ et $b+R\sin(\theta)$, où *theta* est un vecteur allant linéairement d'un angle de départ à un angle d'arrivé. Ainsi, on peut simplement avoir une courbe paramétrique définissant un arc de cercle ou un cercle entier. On modifie la méthode de notre résidu de bord pour notre cercle complet. On a besoin de la normal qui correspond dans notre cas à $nx, ny = (\frac{x}{R}, \frac{y}{R})$. On adapte avec l'aide de la normale nos conditions aux interfaces pour A, B, H.

$$\begin{cases} \Delta A_m = 0 & \in \Omega_m \\ \Delta A_v = 0 & \in \Omega_v \\ (B_{xv} - B_{xm}) \cdot n_x + (B_{yv} - B_{ym}) \cdot n_y = 0 & \in \Gamma \\ (H_{xv} - H_{xm}) \cdot n_y - (H_{yv} - H_{ym}) \cdot n_x = 0 & \in \Gamma \\ A_m - A_v = 0 & \in \Gamma \\ A_v = 0 & \in \delta\Omega \end{cases}$$

On rappelle les relations entre A, B et H:

$$\begin{cases} B_{xv} = \frac{\delta A_v}{\delta y} & B_{yv} = -\frac{\delta A_v}{\delta x} \\ B_{xm} = \frac{\delta A_m}{\delta y} & B_{ym} = -\frac{\delta A_m}{\delta x} \\ H_{xv} = \frac{B_{xv}}{\upsilon_0 \upsilon_r} & H_{yv} = \frac{B_{yv}}{\upsilon_0 \upsilon_r} \\ H_{xm} = \frac{(B_{xm} - B_c)}{\upsilon_0 \upsilon_r} & H_{ym} = \frac{B_{ym}}{\upsilon_0 \upsilon_r} \end{cases}$$

où $\upsilon_0 = 4\pi * 10^{-7}$, $\upsilon_r = 1.01$ et $B_c = 1$. Comme nous avons une division par 10^{-7}, nous donnerons un poids faible pour les H. Même si cela donnait des résultats ressemblant à ceux de Thales. On a choisi d'enlever les constantes multiplicatives, pour éviter des erreurs d'approximations, et de réadapter les poids de la fonction de perte en conséquence. Même si on n'a pas la solution de référence pour comparer, on peut observer que la fonction de perte a du mal à descendre en dessous de 10^{-2}. L'adaptation des poids de la fonction de perte a l'air primordiale et plus délicate que dans les cas précédents. L'algorithme ne converge pas très bien. Même si a vu d'œil, on a eu des solutions ressemblant à ceux de Thales, si on bouge à peine les paramètres, on peut obtenir des choses moins jolie.

Après avoir implémenté ce nouveau problème pour la version normal, on l'implémente en greedy. On se sert du problème de Magnétostatique greedy comme modèle pour l'implémentation. On n'a pas eu tout de suite le fichier de mesures pour ce problème avec un cercle, calculé avec les éléments finis. Donc, dans un premier temps, on compare les résultats à vues d'œil.

Nous avons eu ensuite les solutions calculées avec le logiciel élément fini de Thales pour le cercle, mais aussi des solution plus fine pour le problème de magnétostatique avec rectangle pour plusieurs valeurs de $upsilon_r$ et de Bc. On pourra tester donc avec des valeurs de Bc et $upsilon_r$ différentes, pour mesurer la fiabilité de notre programme sur des problèmes un peu différent.

Nous avons remarqué sur nos entraînements plus que sur les autres la fonction de perte descendait très peu au début de l'entraînement et commencer à réellement descendre à partir d'un certain seuil. Quelques raisons peuvent

expliquer cela, un taux d'apprentissage initial trop bas, une mauvaise régularisation des poids de la fonction de perte, des points de collocations mal choisi, et que le réseau a du mal à apprendre des caractéristiques trop complexe. On va essayer de jouer sur plusieurs de ses points pour améliorer l'entraînement. On augmente tout d'abord le taux d'apprentissage initial de 0.012 à 0.015. La fonction de perte descend déjà mieux.

4.13.1 Correction de l'erreur sur le cercle

Nous avons remarqué que l'erreur avait tendance à se détériorer en avançant dans l'entraînement, et que sur les graphiques d'erreurs, le gros des erreurs avaient l'air placé toujours aux mêmes endroits (bas-gauche, bas-droit, haut-gauche et haut-droit du cercle). Nous avons fait plusieurs modifications des paramètres sans succès. La raison la plus probable de notre problème est dans la fonction de perte de notre bord. J'ai vérifié plusieurs fois en comparaison avec l'article et il semblait qu'elle soit juste. Alors j'ai un peu revu et de comprendre le raisonnement décrit dans l'article. Je pense avoir vu une erreur dans l'article, pour H, on a les conditions : $Hv \times n = Hm \times n$ sur Γ avec n la normal au cercle. Le produit vectoriel nous donne :

$$\begin{pmatrix} H_x \\ H_y \\ 0 \end{pmatrix} \times \begin{pmatrix} n_x \\ n_y \\ 0 \end{pmatrix} = H_x * n_y - H_y * n_x$$

donc on a $H_{vx} * n_y - H_{vy} * n_x = H_{mx} * n_y - H_{my} * n_{mx} \iff (H_{vx} - H_{mx}) * n_y - (H_{vy} - H_{my}) * n_x$. Or dans l'article, on avait un $+$ à la place du moins central.

Après cette correction, les erreurs diminuent mieux, la fonction de perte descend vraiment jusqu'à 10^{-4}, sans qu'on est commencé d'optimiser.

4.14 Récupérer les résultats avec Greedy

Pour accéder aux résultats pour l'algorithme greedy sans passer par l'entraînement, nous avons dû faire quelques modifications de notre programme. Sur l'algorithme classique, nous avons qu'un seul réseau de neurones. Lors de la back-propagation, nous avons besoin de charger un seul réseau et donc d'avoir un seul fichier pour conserver les poids. Pour un réseau Greedy, c'est plus compliqué, pour avoir accès dans la back-propagation, nous avons aussi besoin des réseaux précédents. Nous ne pouvons plus utiliser un seul fichier pour tous les réseaux. Nous créons alors une liste de fichiers. Chaque réseau enregistre ces poids dans un unique fichier. Nous mettons une boucle for pour charger chacun de nos réseaux dans l'ordre en utilisant la fonction déjà implémentée dans scimba. Nous lui avons ajouté un paramètre, pour pouvoir préciser sur quel réseau correspond au fichier que l'on veut charger. Par contre par soucis de simplicité, et pour éviter des erreurs dues à des architectures, des paramètres différents, nous avons dû enlever le chargement des optimiseurs. Cela impacte les mesures d'erreurs, qui ne sont qu'approximativement celle qu'on avait, même si elle reste assez fiable, et

varie peu. La mesure du minimum est la plus impactée, et peut même changer d'ordre de grandeur même si elle reste très basse.

4.15 Reprendre un entraînement

Imaginons que nous sommes satisfaits des résultats de nos deux premiers réseaux de neurones, mais pas du troisième. Ou bien que dans les derniers entraînements, se passe une erreur (par exemple de mémoire). On aimerait bien reprendre au réseau qui a posé problème sans entraîner de nouveau ceux d'avant. Cela est possible, comme l'on a enregistré nos modèles dans des fichiers disjoints. Dans notre fonction training, on charge notre premier modèle, on fait le calcul de notre alpha et nos fréquences, puis on passe au prochain réseau sans refaire l'entraînement, jusqu'au réseau cible, ou on reprend un entraînement normal. Nous remarquons néanmoins quelques problèmes : la fonction de perte n'est qu'approximativement la même et descend moins bien que dans l'entraînement normal. Une hypothèse possible est un problème de schedulder qui part avec un taux d'apprentissage trop grand par rapport à ce qu'il devrait être. Nous avons aussi une petite perte dans les erreurs. D'une mesure à l'autre, nous avons aussi des points de collocations choisis différemment, ce qui peut causer ces petites variations dans l'erreur.

4.16 Letrre Thales

Pendant le stage, j'ai pu échanger par email avec les gens de Thales pour donner les avancer. Nous avons présenté nos résultats de Magnétostatique sur le rectangle et sur le cercle incluant nos meilleures mesures détaillées (pour chaque réseau Greedy et sous-réseau) de la MAE, MSE et l'erreur maximum. Nous avons mis avec cela nos graphiques légendés des différents résidus, erreurs en prenant pour chaque réseau pris en un ensemble et pris séparément.

J'ai pu aussi recevoir de leur part des solutions calculées par éléments finis avec une meilleure précision sur les floatant et une grille plus fine. (On est passé d'environ 40 000 (200*200) à environ 80 000 (285*285) points.)

4.17 Resultat

Pour notre réseau greedy, nous allons présenter nos meilleurs résultats pour chaque problème :

Electrostatique

Notre problème où les fonctions de pertes et mesures descendent le mieux. Les meilleures fonctions de pertes avoisinent les 10^{-11}, l'erreur maximum dans les 10^{-5}, une MAE dans les 10^{-6} et une MSE dans les 10^{-8}. Ces grandeurs sont atteignables en quelques minutes sur gpu. Les graphiques de la fonction de perte nous laissent espérer que si on pousse l'entraînement plus loin dans chaque réseau et avec plus de réseau, nous pourrons avoir de meilleurs résultats.

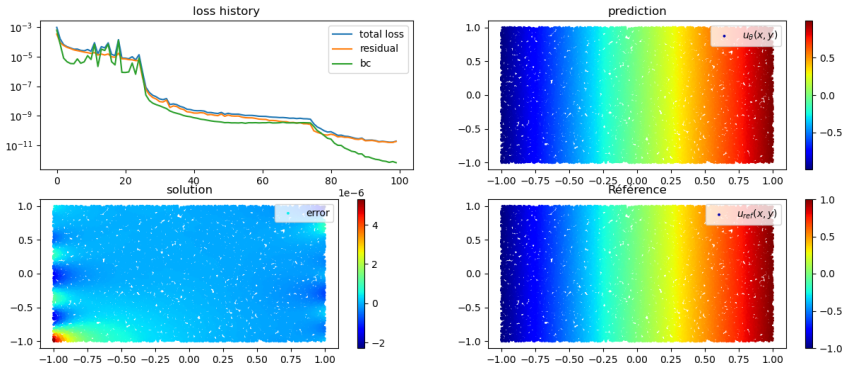

Figure 9: Prédiction du problème d'électrostatique

frequence

Sur notre exemple qui est un laplacien avec des sinus en terme source et qui nous a servis à implémenter les fréquences, nous avons pu atteindre : une erreur maximum en 10^{-3}, une MSE de 10^{-6}, et une MAE de 10^{-4}. Notre erreur est principalement localisée sur les bords. Notre fonction de perte descend dans les 10^{-7}

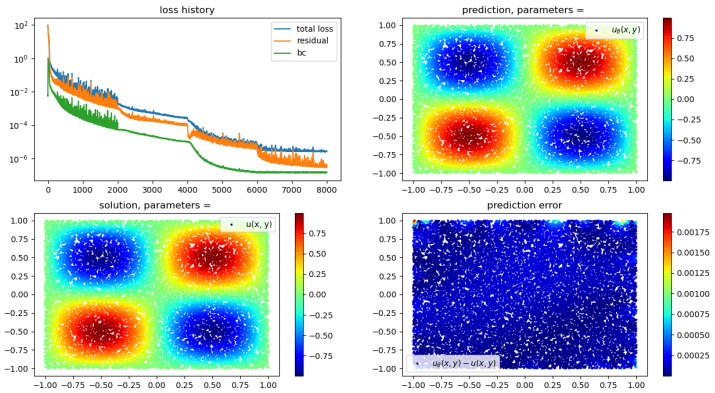

Figure 10: Solution du problème de poisson

Electrostatique avec bimateriaux

Ce problème a été plus compliqué à entraîner en raison des discontinuités à l'interface, et les quatre coins du rectangle où les conditions sont mal définies. L'erreur finale est localisée aux quatre coins du rectangle. La surface où l'erreur est grande a bien diminué par rapport à ce qu'on avait en greedy. L'erreur

maximum reste assez grande, environ 1.10^{-2}. La meilleure MAE est de $1 * 10^{-4}$ et une MSE de $3 * 10^{-6}$. La fonction de perte atteint environ les 10^{-2}. Le temps d'entraînement pour ces résultats peut être approché par 40 minutes sur gpu. La fonction de perte a néanmoins tendance à stagner vers les dernières epochs.

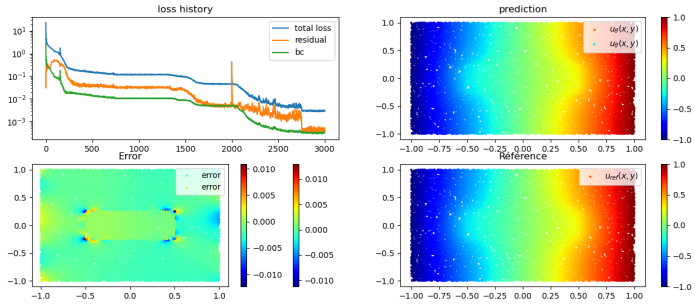

Figure 11: Prédiction du problème d'électrostatique avec deux matériaux

Magnétostatique sur un rectangle

Comme pour le problème précédent, nous avons des discontinuités à l'interface, et une fonction pour le résidu de bord assez complexe. L'erreur final est principalement localisée sur le bord haut et sur le bord bas. Nos meilleurs résultats pour le potentiel sont donnés par une erreur maximum d'environ $2 * 10^{-3}$, une MAE de $3 * 10^{-4}$, une MSE de $3 * 10^{-6}$. Pour Bx, nous avons une erreur maximum d'environ 0.14, une MAE de $3 * 10^{-3}$, une MSE de $1 * 10^{-4}$. Pour By, nous avons une erreur maximum d'environ 0.14, une MAE de $1 * 10^{-3}$, une MSE de $8 * 10^{-5}$. Pour Bx et By, l'erreur est maximum aux environs des quatre coins du rectangle et assez petite ailleurs. Nos résultats sont obtenus après plus d'une heure de gpu. La fonction de perte a tendance à stagner vers les $8 * 10^{-2}$ et osciller assez fortement sur les réseaux de Fourier.

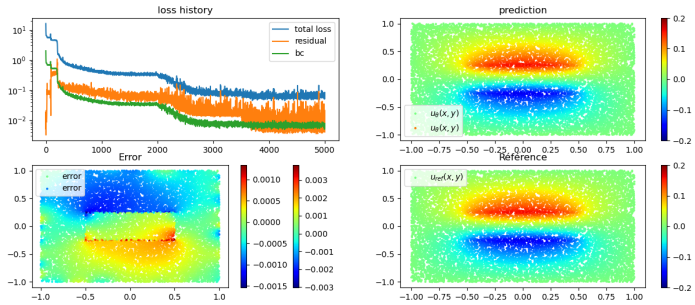

Figure 12: Prédiction du potentiel du problème de magnétostatique sur un rectangle

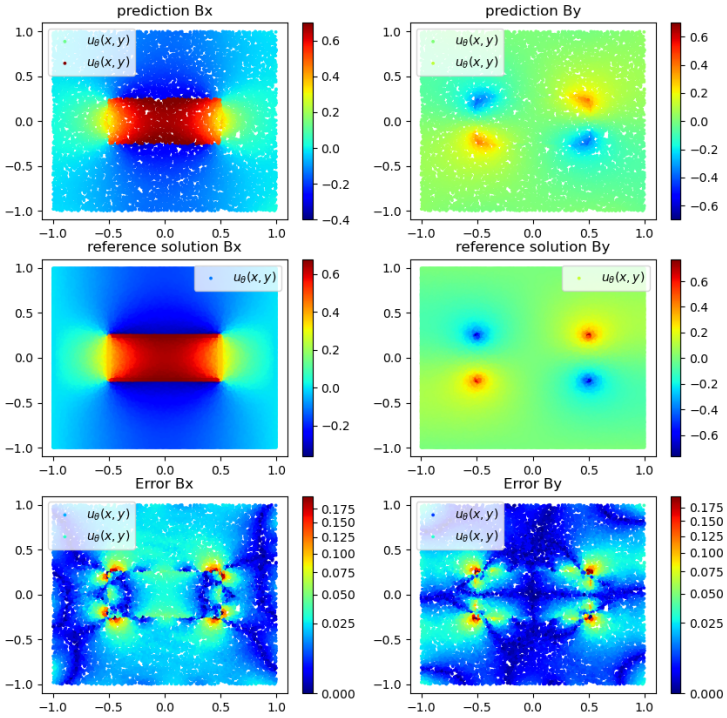

Figure 13: Prédiction Bx, By du problème de magnétostatique sur un rectangle

Magnetostatique sur un cercle

Contrairement aux deux précédents, la fonction de perte descend plutôt bien sur notre problème du cercle. Nous n'avons plus le problème des quatre coins. Cette fonction de perte peut descendre jusqu'à 10^{-10}. Nos meilleurs résultats pour le potentiel sont donnés par une erreur maximum d'environ $9.9 * 10^{-4}$, une MAE de $2 * 10^{-4}$, une MSE de $7 * 10^{-6}$. Pour Bx, nous avons une erreur maximum d'environ 0.02, une MAE de $1 * 10^{-3}$, une MSE de $1 * 10^{-5}$. Pour By, nous avons une erreur maximum d'environ 0.018, une MAE de $5.8 * 10^{-4}$, une MSE de $6 * 10^{-6}$. Pour A, Bx et By, l'erreur est principalement concentré sur le bord du cercle.

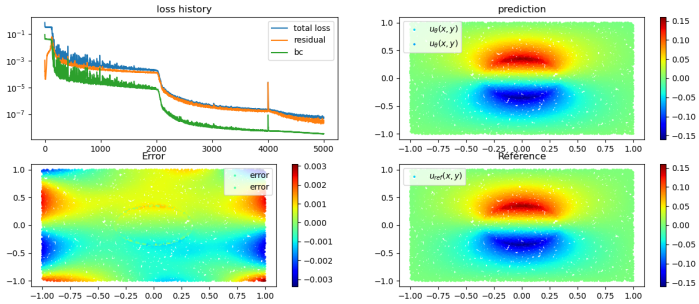

Figure 14: Prédiction du potentiel du problème de magnétostatique sur un cercle

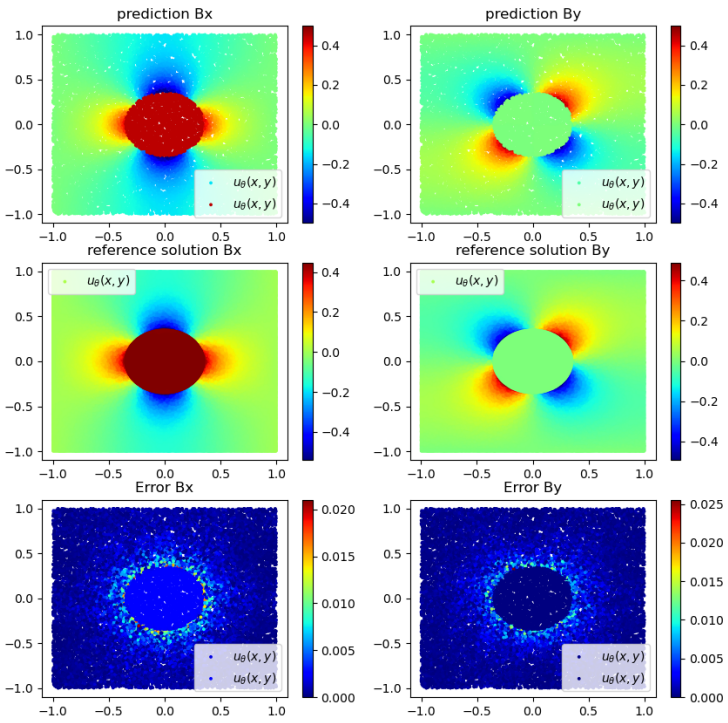

Figure 15: Prédiction Bx, By du problème de magnétostatique sur un

5 Conclusion

En conclusion, nous allons comparer l'utilisation de l'algorithme greedy par rapport à celui de base. On peut déjà remarquer que sur tout nos problèmes, l'erreur s'est améliorée. Le cas d'électrostatique simple est celui dans l'erreur a le mieux diminuer avec une erreur maximum de 10^{-5}. Sur les autres problèmes contenant des discontinuités, et où les normales dans les coins sont mals définis, l'erreur descend moins bien, particulièrement l'erreur maximum qui reste concentrée dans les coins et aux bords des discontinuités. La MSE atteint néanmoins les 10^{-6}.

Le temps que l'algorithme prend augmente beaucoup avec le greedy, surtout sur les derniers réseaux FNN qui peuvent avoir beaucoup de features. Même sur gpu, on peut atteindre quelques heures de complétion. Le nombre de paramètres surtout sur les dernières couches, la complexité de l'entraînement fait qu'il est possible d'avoir plus de problèmes de mémoires avec greedy que sur la normale.

On a alors la nécessité de balancer la complexité de nos paramètres et de la capacités de la machine.

Comme l'algorithme classique, en particulier sur nos problèmes avec un sous-domaine rectangulaire, la fonction de perte et l'erreur ont l'air de stagner à partir d'un certain pallié. Il est assez délicat de passer en dessous de ce pallié. Comme pour l'algorithme classique, on peut modifier certains paramètres ou structures de réseau, permettant d'avoir de petites améliorations sur certains entraînements. Les grandes améliorations sont plutôt difficiles à avoir.

Nous allons faire brièvement le bilan de ce qu'on a pu acquérir leur de ce stage comme compétence. J'ai pu améliorer mes connaissances en machine learning et en particulier pour les PINNs et la résolution d'équation avec des réseaux de neurones.

Concernant les langages de programmation, j'ai pu apprendre quelques spécialités de python (par exemple avec kwargs et les listes d'arguments), d'approfondir les connaissances dans certains modules comme NumPy, mathplotlib (graphique), SciPy (interpolation), Pandas (pour lire des documents csv), et Torch. J'ai pu appliquer et approfondir les connaissances en machine learning et en Torch. J'ai pu étudier la syntaxe de Torch, pouvoir manipuler des éléments du code en torch, comprendre les erreurs de Torch et les résoudre et comprendre les spécificités du module. Comme scimba est un assez grand framework, le stage m'a fait prendre conscience de la nécessité d'organisation des fichiers en module et sous-modules classés par thématique pour pouvoir se retrouver facilement, et éviter des fichiers contenant des milliers de lignes.

Et aussi de devoir factoriser son code, pour éviter si on doit une modification de la faire à plusieurs endroits. J'ai pu aussi approfondir quelques notions de mathématiques comme les fft, la résolution d'équation dans le domaine fréquentiel, les algorithme glouton, quelques faits sur la résolution des équations avec les PINNs. Ces connaissances ont été acquises en partie par la lecture d'article de recherche sur la thématique du stage.

Ce stage m'a aussi appris à organiser la prise de mesures pour les entraînements, la gestion d'un github, où on a stocké les éléments en lien avec le stage (images, mesures, rapport...), et crée des issues... Enfin, pendant ce stage, j'ai pu observer comment fonctionne la communication entre plusieurs acteurs (ici le groupe de Thales avec mes tuteurs de l'IRMA).

6 Annexe

6.1 CNN et PINNs

Nous allons tester dans cette partie un nouveau type de réseau. Nous allons essayer d'implémenter une combinaison de CNN et de MLP. D'après chatgpt d'OpenAI, mettre une couche de CNN pourrait être efficace sur les problèmes avec des discontinuités. Pour implémenter ceci simplement, en prenant en compte les contraintes liées à scimba, nous utilisons Conv2D du module pytorch, et une couche linéaire ayant pour but de redimensionné pour qu'il puisse

passer correctement dans notre double couche GenericMLP de scimba. Une difficulté est la manipulation sur la shape de notre vecteur x. En effet, les réseaux denses et de convolution ne prennent pas exactement la même shape. De plus, on a dû spécifier des paramètres comme le padding, la stride pour que Conv2D garde un vecteur utilisable pour la suite. Nous avons aussi à choisir le nombre de canaux de sortie pour notre couche de convolution. Nous prenons en exemple notre classe `Double_MLP_x` et nous créons notre class `CNN_PINN` en ajoutant notre couche de convolution.

On a essayé plusieurs configurations, mais aucune n'a donné de résultat très probant. Le résidu pouvait descendre assez bas, mais l'erreur de pas beaucoup. De plus, l'entraînement est assez complexe, on utilise plusieurs couches. Notre meilleur résultat obtenu est l'image ci-dessous, où le centre de l'image est beaucoup trop lisse par rapport à ce qu'on voudrait. C'est pourquoi après plusieurs entraînements et quelques modifications du code, on a abandonné l'idée des CNN.

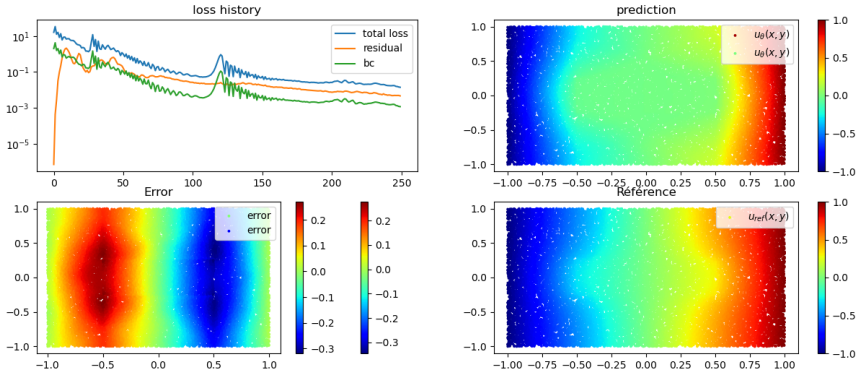

Figure 16: Prédiction sur le problème d'électrostatique avec deux matériaux en utilisant la convolution

6.2 Heavisisde

À la place d'utiliser la combinaison de deux réseaux de neurones, on peut utiliser une fonction d'Heaviside pour gérer nos discontinuités à nos interfaces. Cette idée de fonction de Heaviside est notamment présente dans l'article *Physics-Informed Neural Networks for the Numerical Modeling of Steady-State and Transient Electromagnetic Problems with Discontinuous Media* [3]. On utilisera une approximation continue de la fonction d'Heaviside, avec la fonction sigmoïde. $sigmoid(x,k) = \frac{1}{1+e^{-kx}}$ On utilise notre fonction distance signé pour localisée le bord. A l'intérieur du sous-domaine la fonction est proche de 0, à l'extérieur du sous-domaine la fonction tend vers 1, et à l'interface on a une transition continue entre les deux valeurs. Si on prend (1-H) se sera l'inverse. On fait passer nos points par deux réseaux, on obtient u_pred. On prend alors $H * u\_pred0 + (1-H) * u\_pred1$. Dans la sigmoïde, on peut choisir un

paramètre k qui influe sur le lissage de la discontinuité, avec k grand la pente est très abrupte, avec k petit la pente est plus lissé.

Selon la valeur de k, on peut avoir des résultats peuvent beaucoup variés. Sur les cas utilisant un rectangle, les résultats sans être non plus mauvais ont donné des grandes erreurs à nos quatre coins. En effet, la fonction de Heaviside que l'on utilise n'est pas $\mathcal{C}^\infty$ au partie correspondant au coin du rectangle. Pour ces problèmes, la fonction de Heaviside n'offre pas tellement davantage et détériore les résultats au niveau des dérivées. Le cercle étant continue nous n'avons pas ce genre de problème. Les résultats sur le cercle obtenus sont assez similaires à ceux obtenus sans Heaviside.

Nous avons fait une animation géogébra pour visualiser nos fonctions de Heaviside, dans ce lien https://www.geogebra.org/3d/urppjy7z

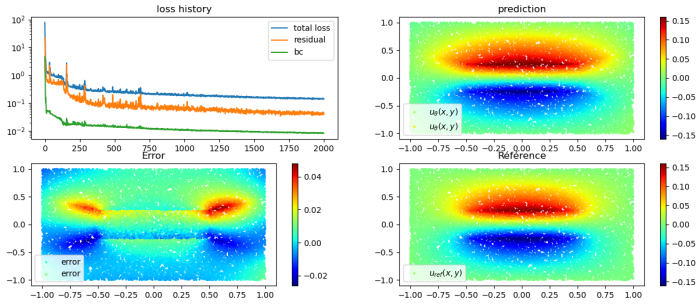

Figure 17: Graphique montrant le type d'erreur que l'on peut avoir si on prend une fonction de Heaviside sur un domaine rectangulaire

Le problème avec le rectangle tel que l'on a implémenté avec Heaviside est qu'il étale les problèmes de discontinuité aux coins du rectangle au lieu de les lisser comme on le voudrait. C'est pourquoi, on a eu l'idée de prendre un rectangle étant arrondi aux coins. La fonction de distance signée est donnée pour ce type de figure par :

$$distanceExt = (max(|x| - (w - r), 0))^2 + (max(|y| - (h - r), 0))^2$$

$$distanceInt = min(max(|x| - (w - r), |y| - (h - r)), 0)$$

$$distance = \sqrt{distanceExt + distanceInt} - r$$

avec w la largeur, h la longueur et r le rayon des coins que l'on peut modifier pour avoir des coins très arrondis ou proche de l'angle droit. On prend ensuite la sigmoïde de cette distance. Sans donner des résultats meilleurs que nos MLP double, le rectangle arrondi fonctionne mieux qu'un rectangle normal. Les résultats sont assez similaires, voire légèrement inférieurs à ceux que l'on obtient avec notre configuration usuelle.

References

[1] Ziad Aldirany, Régis Cottereau, Marc Laforest, and Serge Prudhomme. Multi-level neural networks for accurate solutions of boundary-value problems. *Computer Methods in Applied Mechanics and Engineering*, 419:116666, 2024.

[2] TRYOEN Julie. Mise en œuvre d'un métamodèle pinn pour la simulation de champ magnétique. Master's thesis, INSA Toulouse, 2022-2023.

[3] Michel Nohra and Steven Dufour. Physics-informed neural networks for the numerical modeling of steady-state and transient electromagnetic problems with discontinuous media, 2024.

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

[5] Jonathan W. Siegel, Qingguo Hong, Xianlin Jin, Wenrui Hao, and Jinchao Xu. Greedy training algorithms for neural networks and applications to pdes. *Journal of Computational Physics*, 484:112084, 2023.

[6] Yongji Wang and Ching-Yao Lai. Multi-stage neural networks: Function approximator of machine precision. *Journal of Computational Physics*, 504:112865, 2024.

[7] Zixue Xiang, Wei Peng, Xu Liu, and Wen Yao. Self-adaptive loss balanced physics-informed neural networks. *Neurocomputing*, 496:11–34, 2022.