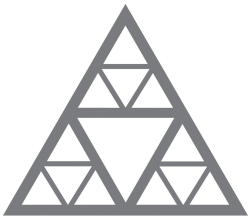


ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES

Physics Informed Neural Networks
Équations d'Helmholtz

Rapport
Projet de Département IMI

Par

Nicolas Beaujoin, Dhia Garbaya, Simon Weissberg

Encadrants :

Antoine Bensalah, Sofiane Haddad, Eric Duceau, Ali Hebbal
Airbus

2024

CONTENTS

1 Motivation

L'apprentissage automatique a déjà démontré des performances dans les domaines de la vision par ordinateur et du traitement. Plus récemment, l'utilisation de ces techniques d'apprentissage pour la résolution des équations différentielles ordinaires (EDO) ou bien des équations aux dérivées partielles (EDPs) a été introduite, notamment par le biais des réseaux de neurones informés physique (PINNs).
Ces méthodes se veulent innovantes et ont pour but de résoudre (et non d'apprendre) des problèmes complexes (non-linéaires, grande dimension...) mais nécessitent souvent la mise en place d'un outillage compliqué sans pour autant avoir les garanties (garantie de convergence, garantie de vitesse de convergence, garantie de bornes d'erreur) qu'offrent les méthodes traditionnelles (éléments finis, différences finies, méthodes spectrales...).
En parallèle, d'autres familles de méthodes ont été proposées permettant l'apprentissage direct d'un opérateur entre espaces de fonctions : on parle alors d'opérateurs neuronaux.
En effet, la plus value des PINNs réside dans le fait qu'on peut les combiner avec d'autres méthodes/formalismes pour apprendre des opérateurs différentiels (opérateurs entre espaces de fonctions). Egalement, les PINNs sont plus flexibles (pas besoin de mesh spécifique) et contrairement aux opérateurs neuronaux, ils peuvent être utilisés sans données.
On propose dans ce projet de se focaliser sur l'équation de Helmholtz en milieu hétérogène – et homogène par morceaux - non borné.

2 Enjeux et État de l'art

2.1 PINNs

Introduits par Maziar Raissi dans [3], les PINN's sont des réseaux de neurones informés par la physique permettant l'approximation de fonctions qui sont solutions d'équations aux dérivées partielles.
Il existe déjà de nombreux réseaux de neurones traditionnels, mais les PINN's méritent le plus grand intérêt pour différentes raisons. En effet, les réseaux traditionnnels ont un grand succès mais restent quand même limités; ils ne prennent pas en compte les lois de la physique, ce que les PINN's font.

Le principe des réseaux de neurones informés par la physique est très simple. Motivé par le succès du deep learning dans des tâches de prédiction ainsi que par la puissance des outils de différentiation automatique (autograd), [3] propose l'idée d'utiliser ces graphes de calcul (réseaux de neurones) afin d'approcher la solution d'une EDP en intégrant dans la fonction objectif du réseau (*loss function*) l'erreur résiduelle de l'EDP et l'erreur résiduelle sur les conditions de bords (cf. partie 3.3).

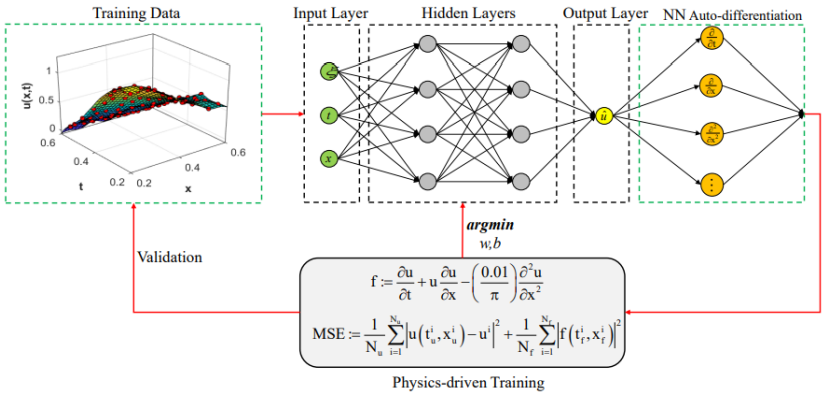


Figure 1: Schéma de l'architecture d'un PINN

Cette idée a priori assez intuitive et simple ne semble pas encore être l'approche magique. En effet, elle est confrontée à certains défis dont on va traiter quelques uns par la suite.

Actuellement, les PINN's sont encore récents et évoluent rapidement grâce à un effort intensif de recherche par la communauté IA pour la physique. Une application réalisée par Ben Moseley a tout de même prouvé leur efficacité sur un problème d'oscillateur harmonique que des réseaux classiques peinaient à approximer. Ils ont aussi été mêlés à des méthodes d'élements finis dites FENA (Finite Element Network Analysis).
Malheureusement, même s'ils sont efficaces sur des problèmes contenant une loi physique quelconque (mécanique, thermique, fluidique, etc..), les PINN's, ou dans ce cas MPINN's (Multiphysics-informed neural network) semblent échouer lorsqu'il y a un couplage de lois, thermique-mécanique par exemple.

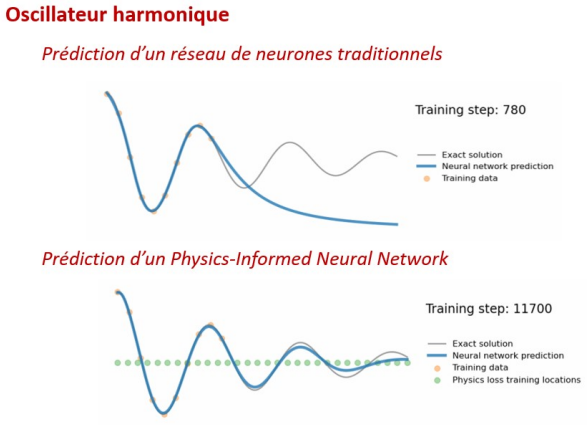


Figure 2: Comparaison entre réseaux traditionnels et PINN's dans le cas de l'oscillateur

2.2 Helmholtz

L'équation de Helmoltz, introduite par le physicien prussien Hermann Von Helmoltz au 19ème siècle, est une équation aux dérivées partielles elliptique apparaissant lors de la recherche de solutions harmoniques de l'équation de propagation des ondes de D'Alembert :

$$-\Delta\varphi(x) - k^2\,\varphi(x) = f(x) \quad \text{in } \Omega$$

Où k est le nombre d'onde et f est un terme source. On recherche donc une certaine fonction φ définie sur un domaine Ω de $\mathbb{R}^n$, et à valeur dans $\mathbb{R}$ ou $\mathbb{C}$. Afin que le problème soit bien posé, on fixe des conditions au bords, celles que nous avons décidé d'étudier sont les conditions de Dirichlet homogènes et non homogènes:

$$\begin{aligned} \varphi &= 0 \quad \text{sur } \partial\Omega \\ \varphi &= \exp\big(-i(k_x x + k_y y)\big) \quad \text{sur } \partial\Omega \end{aligned}$$

Ainsi que des conditions de type guide d'onde avec cette fois-ci $\Omega = [-L, L] \times [-h, h]$:

$$\begin{aligned} &\varphi = 0 && \text{sur } \{x = \pm h\} \cup \{y = L\} \\ &\partial_n \varphi - ik\varphi = 0 && \text{sur } \{y = -L\} \end{aligned}$$

3 Approches et solutions

3.1 Set up général

Dans ce projet, on cherche donc à utiliser les PINNs pour approximer des solutions de différents cas de l'équation de Helmholtz. Comme l'approximation fournie par notre réseau de neurones doit être solution du problème énoncé dans la section 2.2, elle doit vérifier l'équation de Helmholtz et les conditions aux bords, ce qui implique donc l'existence de deux termes dans la loss, un terme dit de loss résiduelle (qu'on notera souvent L_f) lié au fait que la prédiction du réseau de neurones doit vérifier l'équation, et un terme lié aux conditions aux bords (qu'on notera souvent L_b).

Loss Résiduelle: Pour une EDP générale de la forme $\mathcal{N}[u](\mathbf{x}) = 0$ sur un domaine Ω, où $\mathcal{N}$ est un opérateur différentiel et $u(\mathbf{x})$ est la solution, la perte résiduelle L_r est définie comme :

$$L_r = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \mathcal{N}\left[u_\theta\right](\mathbf{x}_i) \right|^2$$

où N_r est le nombre de points de collocation, u_θ est l'approximation par le réseau de neurones de u, et $\mathbf{x}_i \in \Omega$ sont les points de collocation. θ correspond aux paramètres du PINNs que l'on optimise.

Loss aux bords: Pour une condition aux limites de Dirichlet $u(\mathbf{x}) = g(\mathbf{x})$ sur la frontière $\partial\Omega_D$ et une condition aux limites de Neumann $\partial u/\partial n = h(\mathbf{x})$ sur la frontière $\partial\Omega_N$, les pertes aux frontières L_{b_D} et L_{b_N} sont définies comme:

$$\begin{aligned} L_{b_D} &= \frac{1}{N_{b_D}} \sum_{j=1}^{N_{b_D}} \left| u_\theta\left(\mathbf{x}_j\right) - g\left(\mathbf{x}_j\right) \right|^2 \\ L_{b_N} &= \frac{1}{N_{b_N}} \sum_{k=1}^{N_{b_N}} \left| \frac{\partial u_\theta}{\partial n}\left(\mathbf{x}_k\right) - h\left(\mathbf{x}_k\right) \right|^2 \end{aligned}$$

où N_{b_D} et N_{b_N} sont le nombre de points sur les frontières de Dirichlet et de Neumann respectivement, et $\mathbf{x}_j \in \partial\Omega_D$ et $\mathbf{x}_k \in \partial\Omega_N$ sont les points aux frontières.

Dans nos expérimentations, on cherche à minimiser une loss totale de la forme $L_f + L_b$ ou bien une combinaison linéaire de ces deux loss. Mais on peut en fait utiliser une autre méthode

lorsque l'on a des conditions aux bords de type Dirichlet : on peut intégrer les conditions aux bords dans une seule loss. Pour faire ceci, on modifie la sortie (la méthode forward) du réseau de neurones. Par exemple, dans le cas de l'équation de Helmholtz en une dimension avec une condition au bord de type Dirichlet (par exemple $u_\theta(0) = 0$), le réseau va retourner : $\psi_\theta(x) = x \times u_\theta(x)$, où u_θ vérifie l'équation de Helmholtz, et donc ψ_θ vérifie bien la condition au bord.

3.2 Etude de fréquence

3.2.1 En théorie

Essentiellement, un réseau de neurones va d'abord apprendre les données associées à des fonctions de basse fréquence, puis ajoute progressivement des fréquences plus élevées pour améliorer l'ajustement. Ceci est illustré sur la Figure 2, issue de [4].

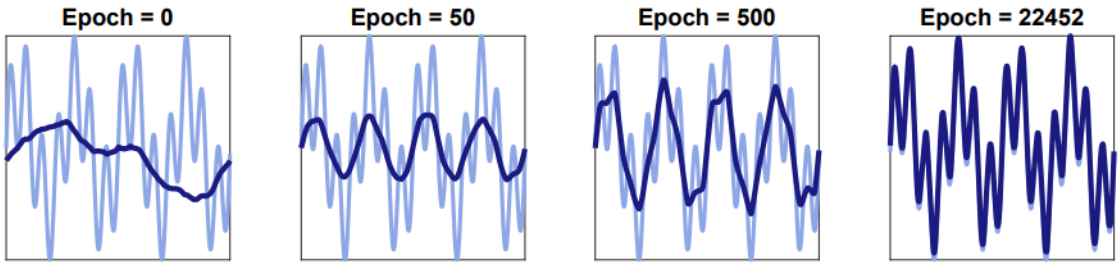


Figure 3: *Prédiction du réseau (bleu foncé) pour une superposition de deux ondes sinusoïdales de fréquences k = 4, 14 (bleu clair). Le réseau ajuste la composante de basse fréquence de la fonction après 50 époques, alors qu'il n'ajuste la fonction complète qu'après environ 22K époques.*

L'article [4] s'appuie sur une approche d'analyse de fréquence pour étudier la dynamique d'apprentissage des réseaux de neurones entraînés par descente de gradient. [4] met en évidence la façon dont ils apprennent les fonctions de basse fréquence plus facilement que celles de haute fréquence.

Les principales conclusions sont que les réseaux de neurones évitent l'overfitting malgré leur grand nombre de paramètres, car ils ont tendance à explorer et à se fixer sur des solutions plus simples et à basse fréquence dès le début du processus d'apprentissage. L'étude montre que les réseaux de neurones apprennent les composantes de basse fréquence d'une fonction beaucoup plus rapidement que les composantes de haute fréquence. Ceci est basé sur la relation entre la fonction à apprendre et les vecteurs propres et valeurs propres d'une matrice spécifique H^∞. Les entrées de cette matrice sont données par :

$$H_{ij}^\infty = \frac{1}{2\pi}\mathbf{x}_i^T\mathbf{x}_j\left(\pi - \arccos\left(\mathbf{x}_i^T\mathbf{x}_j\right)\right)$$

Où les $\mathbf{x}_i$ sont les données. L'analyse indique que pour les fonctions unidimensionnelles, le temps d'apprentissage est proportionnel au carré de la fréquence k de la fonction.

Les expériences confirment que les prédictions théoriques sont valables non seulement pour des modèles simples de réseaux, mais aussi pour des réseaux profonds plus réalistes. Cela

est démontré dans des scénarios où le réseau interpole les données manquantes à l'aide de fonctions à basse fréquence, évitant ainsi un surajustement.

Le fait que les réseaux de neurones donnent naturellement la priorité à l'apprentissage de fonctions de basse fréquence contribue à leurs solides capacités de généralisation. Ce processus d'apprentissage sélectif explique également pourquoi les modèles surparamétrés ne conduisent pas nécessairement à un surajustement.

3.2.2 Une solution : *Fourier features*

Nous pouvons approximer les réseaux de neurones profonds avec la régression par noyau : utiliser un réseau neuronal profond équivaut à faire une régression avec le noyau tangent neuronal (NTK). La dynamique de la fonction f_θ du réseau suit celle de la descente du gradient du noyau dans l'espace des fonctions par rapport à un noyau limitatif. [4]

Les MLP (Multi Layer Perceptrons) standards correspondent à des noyaux dont la chute de fréquence est rapide, ce qui les empêche de représenter le contenu à haute fréquence présent dans les images et les scènes naturelles. [5]

Pour remédier à cette limitation, [5] introduit une technique appelée "Fourier feature mapping". Cette méthode consiste à transformer les coordonnées d'entrée à l'aide de fonctions sinusoïdales avant de les faire passer par le MLP. Cette transformation modifie le noyau neuronal tangent (NTK) associé au MLP, le rendant stationnaire et permettant de contrôler la gamme de fréquences que le MLP peut apprendre en ajustant les vecteurs de fréquence. Les résultats empiriques montrent que cette technique améliore de manière significative les performances du MLP dans l'apprentissage de fréquences plus élevées, améliorant ainsi des tâches telles que l'inférence de formes et la synthèse de nouvelles vues.

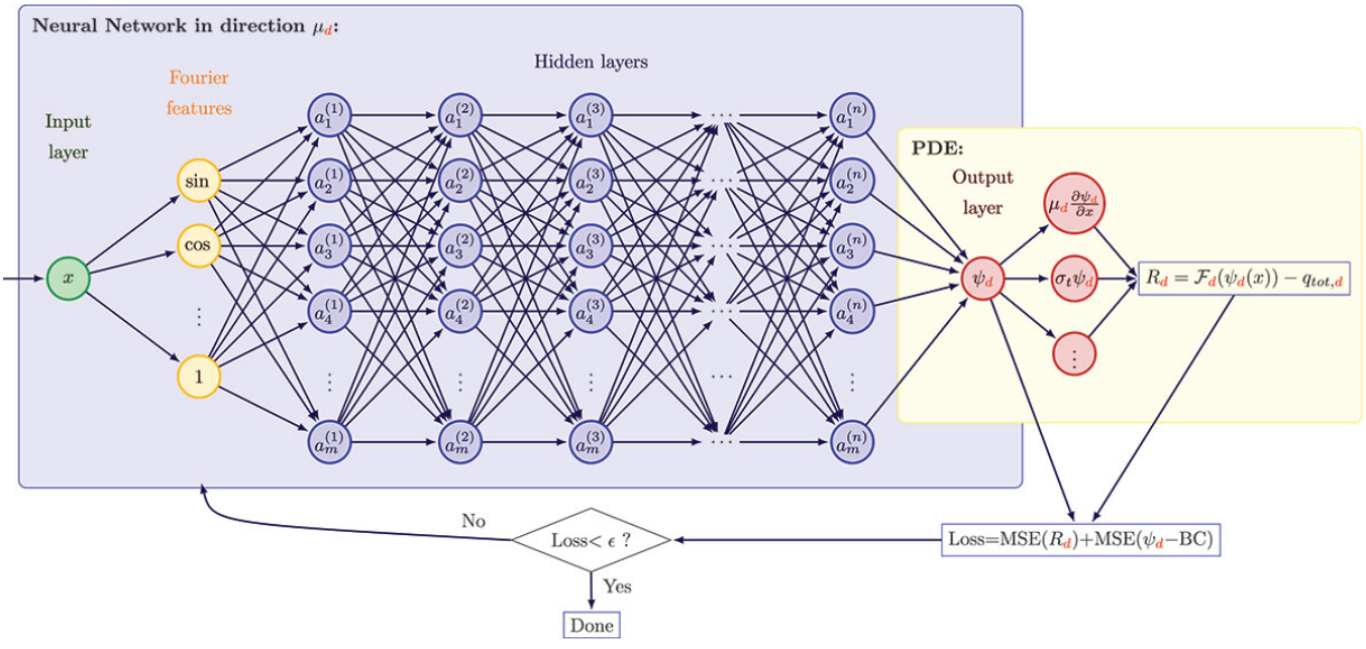


Figure 4: Représentation schématique de l'utilisation des *Fourier features*.

Fourier features :

Points en entrées : $v \in [0,1]^d$.

$$\gamma(v) = [a_1 \cos(2\pi b_1^T v), a_1 \sin(2\pi b_1^T v), a_2 \cos(2\pi b_2^T v), a_2 \sin(2\pi b_2^T v), ..., a_m \cos(2\pi b_m^T v), a_m \sin(2\pi b_m^T v)].$$

Où les $a_1, a_2, ..., a_m, b_1, b_2, ..., b_m$ sont des paramètres ajustables. Plus particulièrement, les a_i sont des coefficients réels et les b_i sont des vecteurs.

La fonction noyau induite par cette fonction (*Fourier features*) est stationnaire (elle ne dépend que de la différence entre les points). Ensuite il reste seulement à passer les points $\gamma(v)$ dans le réseau de neurones.

En manipulant les paramètres a_j et b_j, il est possible de modifier radicalement le taux de convergence et le comportement de généralisation du réseau résultant. [5] montre que la stratégie simple consistant à fixer $a_j = 1$ et à échantillonner aléatoirement b_j permet d'obtenir de bonnes performances.

Les *Fourier Features* peuvent rendre les MLP mieux adaptés à la modélisation de fonctions, surmontant ainsi le biais spectral inhérent aux MLP. [5] a montré expérimentalement que le réglage des paramètres des *Fourier Features* permet de contrôler la chute de fréquence du NTK et d'améliorer significativement les performances.

3.2.3 Cas d'études

On cherche dans cette section à étudier les phénomènes énoncés dans la section précédente, dans le cas des **Physics Informed Neural Networks** utilisés pour approximer des solutions de l'**équation de Helmholtz**. On se place dans un cadre assez simple où l'on cherche à approximer une solution analytique de la forme :

$$u(x, y) = \sin(k_x^* x) \sin(k_y^* y)$$

On se place dans le domaine $[-1, 1] \times [-1, 1]$. On a un terme source de la forme : $f(x, y) = (k^2 - k_x^{*2} - k_y^{*2}) \sin(k_x^* x) \sin(k_y^* y)$
L'équation vérifiée par la solution u est :

$$\Delta u + k^2 u = f$$

Avec comme conditions aux bords (Dirichlet homogènes):

$$u(-1, y) = u(1, y) = u(x, -1) = u(x, 1) = 0$$

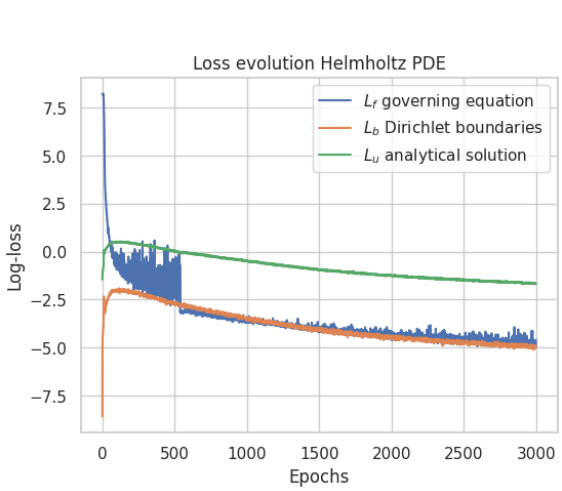
On fixe ici la valeur de k à $k = 2\pi$ et on va faire varier les valeurs de k_x^* et k_y^*.

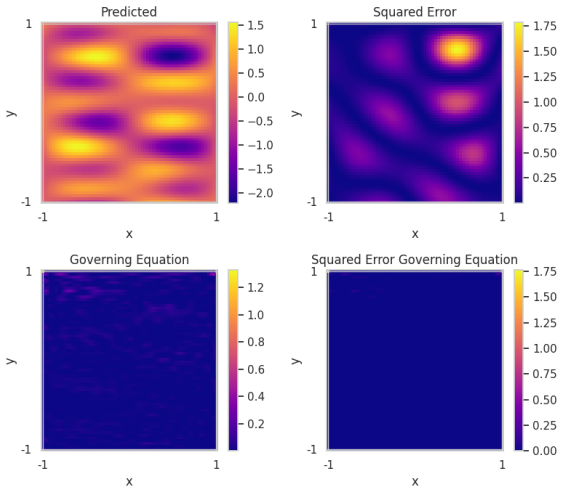
Les tests dans cette section sont effectués sous Python avec tensorflow, avec un réseau de neurones ayant 3 couches denses (linéaire, activation) avec 64 unités chacune. On choisit une learning rate valant 0.001 et avec ReduceLROnPlateau (patience de 100, cette fonction permet de réduire la learning rate lorsque la loss ne diminue pas pendant plusieurs époques, i.e. on est sur un plateau). On effectue un entraînement pendant 3000 époques. Les 4000 points de collocation dans le domaine sont générés aléatoirement (random uniform) à chaque époque, et on en prend 100 sur chaque segment du bord (soit 400 points sur le bord au total). Cette méthode est donc bien différente des méthodes d'approximation traditionnelles, où on avait plutôt tendance à utiliser un maillage régulier fixé pour tout l'apprentissage.

Dans ce qui suit, le graphe "Predicted" correspond à la prédiction donnée par le réseau de neurones, "Squared Error" correspond à une approximation de la norme L^2 de l'écart entre la solution analytique et la prédiction du réseau de neurones, "Governing Equation" correspond à l'erreur résiduelle de l'équation, et "Squared Error Governing Equation" correspond à la même erreur au élevée au carré. De plus la fonction de loss L_f "governing equation" correspond à la loss résiduelle de l'EDP, L_b "Dirichlet boundaries" correspond à la loss sur les points des bords et L_u "analytical solution" correspond à l'écart entre la solution analytique et la solution prédite par le réseau de neurones. On précise que lors du processus d'entraînement, c'est la somme $L_f + L_b$ (ou une combinaison linéaire de ces deux loss) qui est minimisée, et non pas L_u. L_u est juste calculée au cours des époques à titre indicatif, pour voir si la solution prédite par le réseau se rapproche ou non de la solution analytique.

3.2.4 Dans le cas $k_x^* = \pi$ et $k_y^* = 4\pi$

On fait un premier test avec ces valeurs de k_x^* et k_y^*.





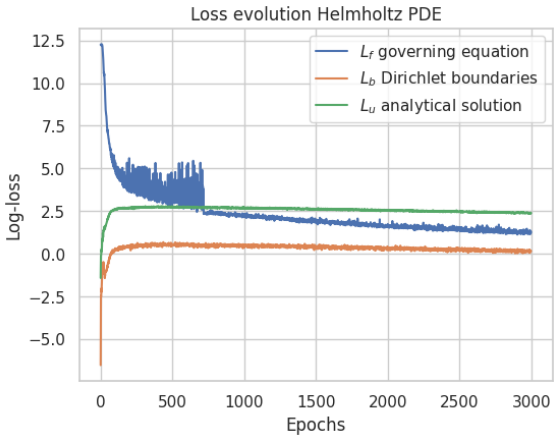
On reconnaît sur le graphe "Predicted" la forme de la solution attendue. De plus, les fonctions de loss (résiduelle et bords) semblent converger vers 10^{-5} au bout de 3000 époques. On peut en déduire que notre PINN approxime bien la solution à notre équation. On remarque tout de même sur le graphe "Squared Error" plusieurs lobes en lesquels la solution analytique semble être moins bien approximée. Ceci est peut-être dû au fait qu'il y a des zones du domaine où il y a moins de points de collocation qu'à d'autres endroits. On peut permuter les valeurs de k_x^* et k_y^* pour vérifier que le réseau de neurones se comporte de la même manière dans les deux directions, en prenant $k_x^* = 4\pi$ et $k_y^* = \pi$. En effectuant le même test, on obtient des résultats similaires (même décroissance des fonctions de loss et même graphe "Predicted", à transposition près).

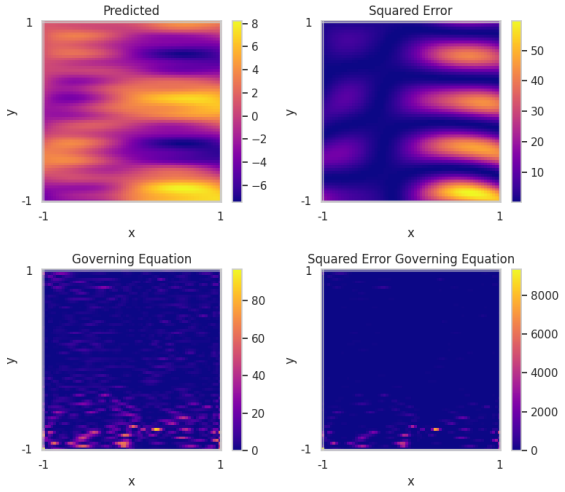
Dans les cas de basses fréquences $k_x^* = 2\pi$ et $k_y^* = 3\pi$, $k_x^* = \pi$ et $k_y^* = 2\pi$, on obtient également des résultats satisfaisants de convergence.

Jusqu'à présent notre processus d'apprentissage visait à minimiser une loss totale égale à $L_f + L_b$ où L_f est la loss résiduelle et L_b la loss sur les bords. Mais au vu des valeurs qu'on obtient pour L_f au cours de l'apprentissage, on peut plutôt chercher à minimiser une loss

totale de la forme $10^{-3} \times L_f + L_b$. Ce qui permet en fait de rééquilibrer l'importance donnée à chacune des deux loss. La section 3.3 étudiera plus en détail l'équilibrage des loss.

3.2.5 Dans le cas $k_x^* = \pi$ et $k_y^* = 10\pi$





Ici, on a largement augmenté la fréquence. Les paramètres utilisés pour l'entraînement sont les mêmes que pour le cas précédent. On remarque premièrement sur le graphe "Predicted" que la solution n'est pas bien approximée. Cependant, la moitié gauche de la fonction semble être un peu mieux approximée que la partie droite. De plus, la loss L_f diminue un peu au cours des époques mais ne converge pas vers une valeur satisfaisante (convergence vers 10 comparé à 10^{-5} du cas précédent). La loss aux bords reste quant à elle presque constante au cours des époques. Il semble donc que le réseau de neurones apprend donc moins bien cette fonction qui a une fréquence plus élevée.

3.2.6 Superposition de produits de sinus

On cherche à approximer une solution de la forme :

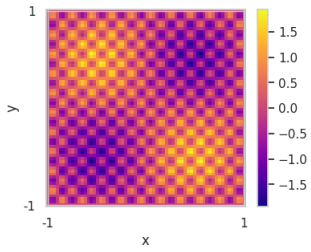
$$u(x, y) = \sin(k_{1x}^* x)\sin(k_{1y}^* y) + \sin(k_{2x}^* x)\sin(k_{2y}^* y)$$

On a donc cette fois un terme source de la forme : $f(x, y) = (k^2 - k_{1x}^{*2} - k_{1y}^{*2})\sin(k_{1y}^* x)\sin(k_{1y}^* y) + (k^2 - k_{2x}^{*2} - k_{2y}^{*2})\sin(k_{2y}^* x)\sin(k_{2y}^* y)$

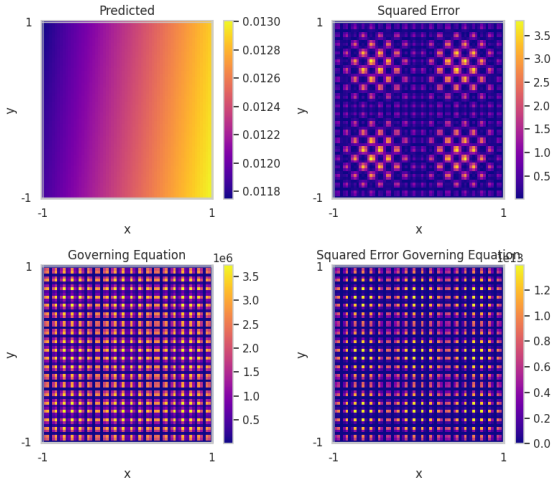
L'équation et les conditions aux bors sont toujours les mêmes que dans la section 3.2.3. On cherche à approximer une superposition de deux produits de sinus, un de basse fréquence et un de haute fréquence.

3.2.7 Test avec $k_{1x}^* = \pi$, $k_{1y}^* = \pi$, $k_{2x}^* = 10\pi$ et $k_{2y}^* = 10\pi$

La solution analytique a l'allure suivante :



On a les résultats suivants :



On remarque donc qu'ici, le réseau n'a pas du tout réussi à approximer la solution.

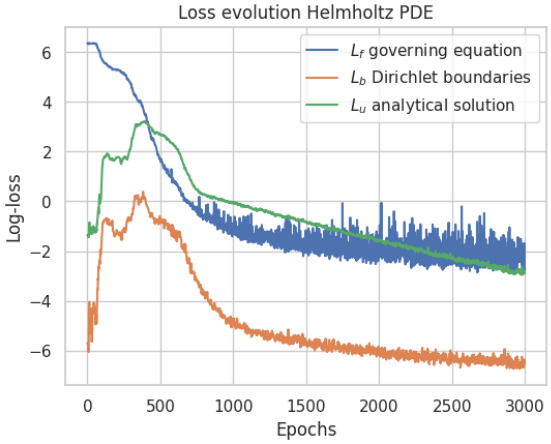
3.2.8 Autres essais

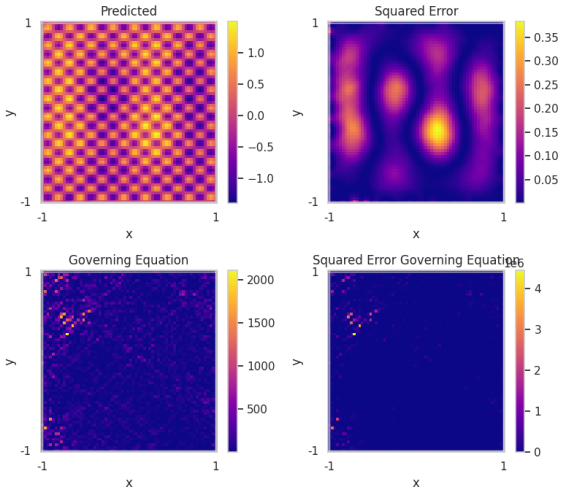
On a essayé plusieurs autres paramétrages sur différents tests.

- Changer les conditions aux bords comme précisé dans la partie 2.2, pour la partie Dirichlet non homogène, la fonction cible φ devient complexe ce qui a entraîné des problèmes, le code n'était pas concluant.
- Utilisation d'une grille régulière uniforme pour la répartition des points de collocation: les résultats des tests effectués avec cette méthode sont beaucoup moins concluants, même pour des solutions de basses fréquences.
- Visualisation d'autres normes (L^1, L^∞) pour étudier l'écart entre la solution analytique et la prédiciton au cours des époques. Cet ajout n'a pas apporté d'information intéressantes car les courbes de ces normes avaient des allures similaires à celle de la norme L^2.
- Utiliser d'autres architectures de réseaux de neurones : on a par exemple fait des tests avec des réseaux plus profonds et moins larges, avec par exemple 5 couches denses contenant 5 unités chacune. Les résultats étaient moins concluants avec cette méthode.
- Faire quelques descentes de gradients avec le même jeu de points de collocation avant de retirer un nouvel échantillon de points de collocation. Cette méthode semble mieux fonctionner que la méthode consistant à utiliser un nouvel ensemble de points de collocation à chaque époque. Par exemple, dans le cas $k_x^* = 8\pi$ et $k_y^* = 10\pi$, lorsque l'on change de training batch à chaque époque, on obtient que le réseau n'a pas du tout

réussi à apprendre la solution.

Dans le cas où l'on change le training batch toutes les 50 époques, on obtient :





Le réseau a donc réussi à apprendre la solution qui a une fréquence élevée. Les valeurs de loss finales sont plutôt satisfaisantes.

- Tester un "cheminement" de coefficients de l'équation de "l'uniforme" vers ceux que l'on veut. Par exemple, augmenter la fréquence lentement vers la fréquence cible, au fur et à mesure de l'apprentissage. Ne pas attendre d'être complètement à convergence avant d'avancer dans le cheminement. Cette méthode est assez efficace pour apprendre des fonctions de hautes fréquences, on a obtenu des résultats satisfaisants en faisant par exemple le cheminement suivant : $(k_x^* = 2\pi, k_y^* = 4\pi) \longrightarrow (k_x^* = 4\pi, k_y^* = 6\pi) \longrightarrow (k_x^* = 8\pi, k_y^* = 10\pi)$.

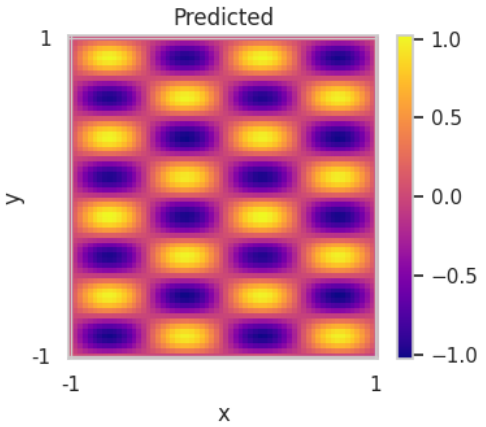


Figure 5: Résultat pour $(k_x^* = 2\pi, k_y^* = 4\pi)$ au bout de 800 époques

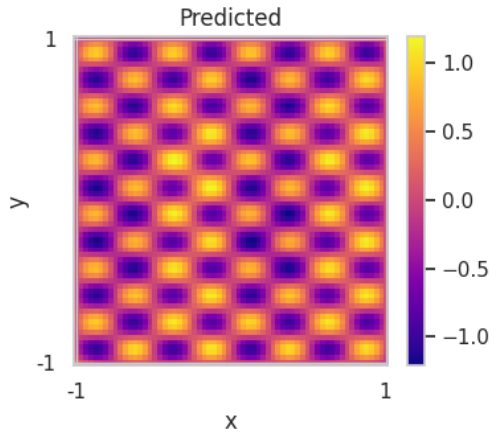


Figure 6: Résultat pour $(k_x^* = 4\pi, k_y^* = 6\pi)$ au bout de 500 époques avec initialisation avec les coefficients de l'entraînement précédent

Autres idées non essayées :

- Tirer les points de collocation selon un LHS (Latin Hypercube Sampling) randomisé (on tire des points uniformément dans les cases d'un quadrillage donnée à l'avance). Cela permettrait d'éviter les effets de "paquets de points" / "trous sans point" dans le domaine dû à l'échantillonnage uniforme.

3.2.9 Test des *Fourier Features*

Ici, on a des points qui sont dans le domaine $[-1, 1]^2$.
On utilise donc une fonction de la forme :
$\gamma(v) = [a_1 \cos(2\pi b_1^T v), a_1 \sin(2\pi b_1^T v), a_2 \cos(2\pi b_2^T v), a_2 \sin(2\pi b_2^T v), ..., a_m \cos(2\pi b_m^T v), a_m \sin(2\pi b_m^T v)]$.

Le code utilisé dans cette section pour implémenter les Fourier Features est celui utilisé dans [5]. On n'a pas réussi à obtenir de résultats concluants dans cette section, les temps d'entraînements étaient trop longs et lorsque l'on mettait peu d'époques, on n'obtenait pas de bons résultats.

3.2.10 Synthèse des résultats

On présente dans ce tableau les valeurs des loss à la fin de l'entraînement (au bout des 3000 époques). Les conditions des tests sont les mêmes que celles énoncées dans la section 3.2.3.

	$(\pi, 4\pi)$	$(4\pi, \pi)$	$(2\pi, 3\pi)$	$(\pi, 2\pi)$	$(\pi, 8\pi)$	(π, π)	$(\pi, 10\pi)$	$(10\pi, \pi)$
L_f	10^{-5}	10^{-5}	10^{-6}	10^{-7}	1	10^{-8}	10^{1}	1
L_b	10^{-5}	10^{-5}	10^{-6}	10^{-5}	$10^{-2.5}$	10^{-7}	1	10^{-1}

3.3 Equilibrage des loss

Pour rappel, les principales composantes de la fonction loss (objectif) dans un réseau de neurones informés par la physique (PINN) sont :

- La loss Résiduelle : Mesurant la déviation de la sortie du réseau de neurones par rapport à la satisfaction de l'EDP aux points de collocation.
- La loss aux bords (simple ou composée): Motivant la satisfaction des conditions aux limites, assurant que la solution respecte au mieux les contraintes connues aux frontières du domaine.

3.3.1 Nécessité

L'équilibrage efficace de ces pertes **de natures différentes** est crucial pour la stabilité et la convergence du PINN, car il s'agit d'un problème multi-objectifs nécessitant donc au moins une scalarisation (i.e pondération). En effet, la loss résiduelle, dans une équation d'Helmholtz, fait intervenir le laplacien de la solution prédite qui s'avère avoir en général un ordre de grandeur plus important que celui de la solution elle même ou de ses dérivées premières. D'où l'importance de la mise en échelle qui évite la dominance d'une seule loss.

3.3.2 Méthodes

- Exemple d'Équilibrage Statique:
  Supposons que nous avons une fonction de perte totale L définie par:

$$L = \lambda_r L_r + \lambda_{b_D} L_{b_D} + \lambda_{b_N} L_{b_N}$$

Un choix possible peut être: $\lambda_{b_D} = \lambda_{b_N} = 10^4$ et $\lambda_r = 1$.

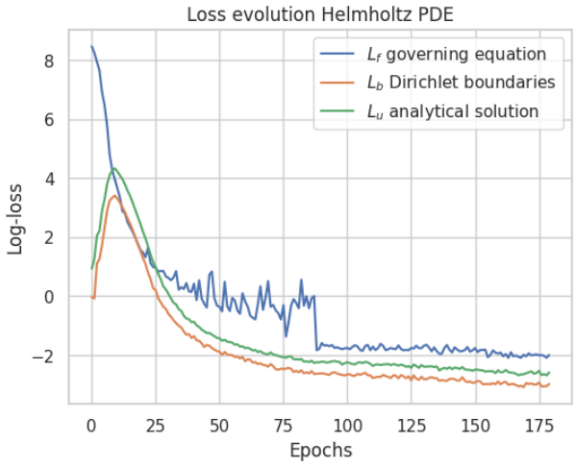
- Stratégie d'Équilibrage Dynamique:
  - Pondération basée sur les gradients: Cette stratégie utilise les gradients des composantes de perte pour ajuster les poids. La perte totale peut être équilibrée dynamiquement par :

$$\lambda_r(t) = \frac{\|\nabla_\theta L_{b_D}(t)\| + \|\nabla_\theta L_{b_N}(t)\|}{\|\nabla_\theta L_r(t)\|}$$
$$\lambda_{b_D}(t) = \frac{\|\nabla_\theta L_r(t)\| + \|\nabla_\theta L_{b_N}(t)\|}{\|\nabla_\theta L_{b_D}(t)\|}$$
$$\lambda_{b_N}(t) = \frac{\|\nabla_\theta L_r(t)\| + \|\nabla_\theta L_{b_D}(t)\|}{\|\nabla_\theta L_{b_N}(t)\|}$$

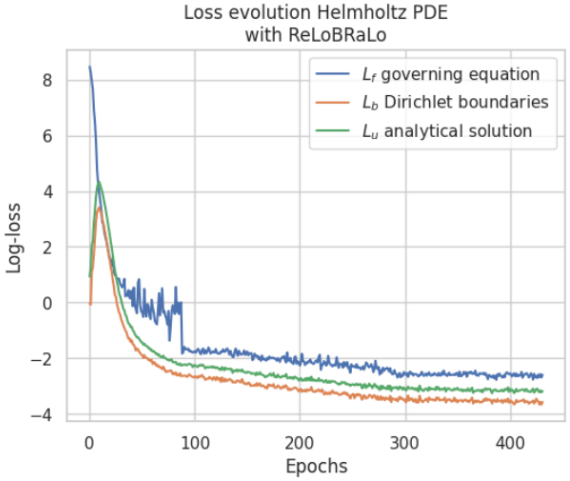
  où $\|\nabla_\theta L_i(t)\|$ désigne la norme du gradient de la composante de perte L_i par rapport aux paramètres du réseau θ à l'instant t.
  - Annealing: où les coefficients de pondération sont ajustés progressivement pour équilibrer les contributions des différentes pertes. Par exemple, les poids de contributions peuvent varier selon une fonction décroissante de température.

Figure 7: Évolution des coefficients des loss sur un exemple d'équation de Helmholtz [1].

- **ReLoBRaLo (Random Lookbacks, Annealing)**: Cette méthode proposée dans [1] combine des retours en arrière aléatoires avec un processus d'annealing progressif pour ajuster dynamiquement les poids des loss, améliorant ainsi l'équilibre et la convergence. Elle s'adapte aux variations des contributions des différentes pertes au cours de l'entraînement.



(a) Équilibrage statique



(b) Équilibrage dynamique

On remarque une convergence globalement meilleure et légèrement plus rapide.

3.4 Cas complexe dans un guide d'onde:

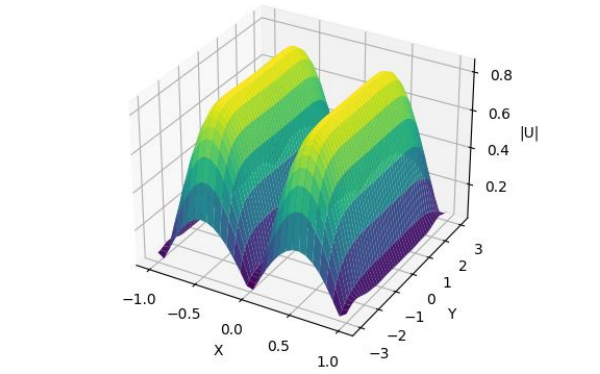
On considère dans ce paragraphe le cas d'un guide d'onde décrit dans l'introduction 2.2. Dans une fonction physics-loss, nous implémentons le calcul du laplacien, de dérivées normales ainsi que des trois losses mises en jeu (résiduelle, dirichlet, rubin).

```
    predictions = model(X)     # (N_coloc_points, 2)
    x = X[:,0].unsqueeze(1)  # make it a column vector
    y = X[:,1].unsqueeze(1)

    # Extract real and imaginary parts of predictions
    u_real, u_imag = torch.chunk(predictions, 2, dim=1)

    # Compute first derivatives w.r.t. each component of X
    grad_u_real = grad(u_real, X, grad_outputs=torch.ones_like(u_real),
        create_graph=True)[0]
    grad_u_imag = grad(u_imag, X, grad_outputs=torch.ones_like(u_imag),
        create_graph=True)[0]

    # Extract components of gradients
    dudx, dudy = grad_u_real[:, 0], grad_u_real[:, 1]
    dvdx, dvdy = grad_u_imag[:, 0], grad_u_imag[:, 1]

    # Compute second derivatives for the Laplacian
    dud2x = grad(dudx, X, grad_outputs=torch.ones_like(dudx), create_graph
        =True)[0][:, 0]
    dud2y = grad(dudy, X, grad_outputs=torch.ones_like(dudy), create_graph
        =True)[0][:, 1]
    dvd2x = grad(dvdx, X, grad_outputs=torch.ones_like(dvdx), create_graph
        =True)[0][:, 0]
    dvd2y = grad(dvdy, X, grad_outputs=torch.ones_like(dvdy), create_graph
        =True)[0][:, 1]

    laplacian_u = dud2x + dud2y
    laplacian_v = dvd2x + dvd2y
    laplacian = laplacian_u + 1j * laplacian_v

    normal_derivatives = ( -dudy*(y==-h) + dudy*(y==h)) +
                         1j*(-dvdy*(y==-h) + dvdy*(y==h) )
```

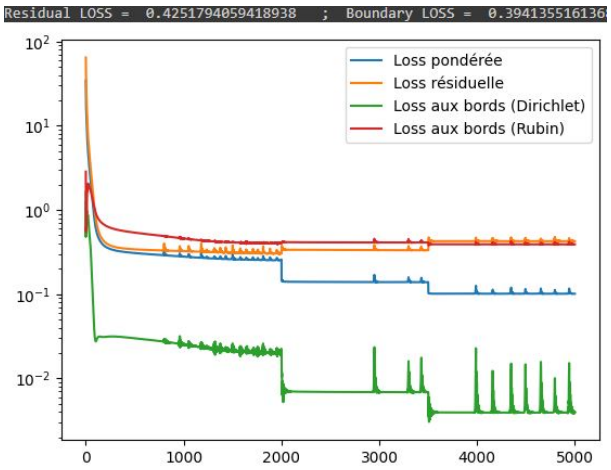
Listing 1: Extrait de la fonction Physics-loss

Pour un choix de Tanh comme activation, un domaine [-1,1]x[-3,3] avec un mesh uniforme de 2700 (8100 points donnent un résultat identique), un réseau de 4 couches de largeur 128 et un entraînemnt de 4000 époques en full-batch, nous avons obtenu les résultats suivants:

La stratégie était de prioriser la loss résiduelle (plus difficile à optimiser) et donner ensuite, sur 2 étapes, de plus en plus d'importance aux loss aux bords.
On remarque qu'on a réussi à capturer l'aspect général de la solution exacte (cf. section 4) mais on a échoué à bien prédire la sinusoïde sur les bords de Robin.



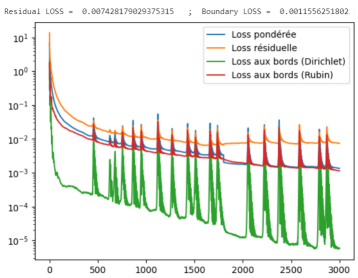
(a) Module de la prédiction



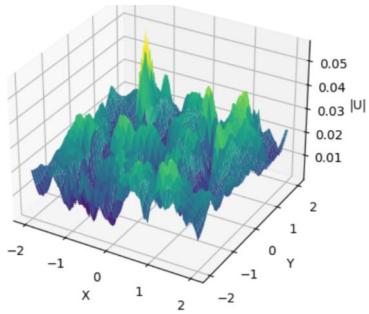
(b) Evolution des fonctions de loss

4 Enjeux et Ouverture

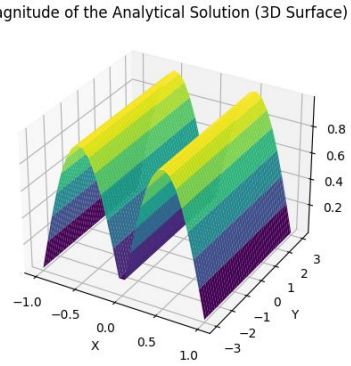
- **Une évaluation plus difficile** : Des valeurs relativement très petites de loss peuvent cependant donner une solution prédite mauvaise. Ceci est dû à la relativité de l'interprétation des valeurs de la loss résiduelle et des loss aux bords de type autre que Dirichlet (faisant intervenir des dérivées) selon les cas. Au contraire des loss classiquement manipulées en machine learning et qui sont data-driven dans un contexte d'apprentissage supervisé. C'était le cas avec l'équation d'Helmholtz dans un guide d'onde borné. Pour un mesh de 8100 points, un domaine de [-2,2]x[-2,2], le choix suivant des activations: [Tanh, ReLU, Sigmoid, Tanh], nous avons réussi à bien optimiser les loss. Néanmoins, la solution prédite, bien qu'elle a réussi à échapper à la solution nulle, est loin d'être bonne. Pour remédier partiellement à cette difficulté, nous proposons de faire des tests sur des échantillons de points différents ou d'opter pour un échantillonage aléatoire durant l'enraînement.



(a) Évolution des losses



(b) Solution prédite

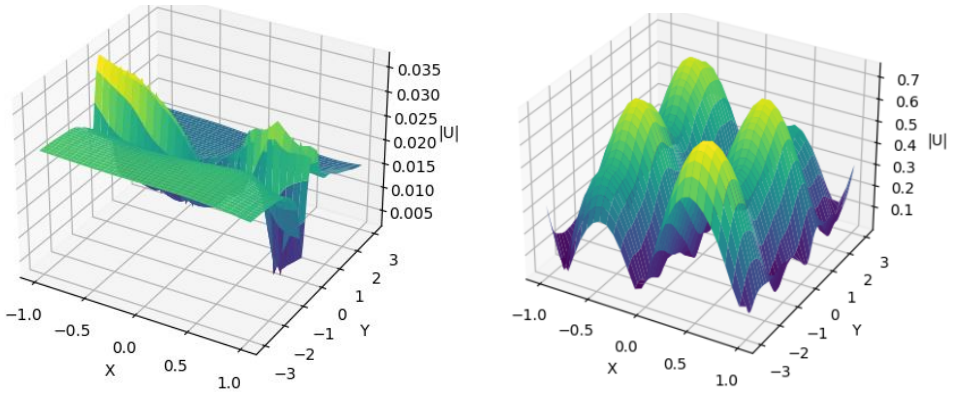


(c) Solution exacte

- **Achitectures :** Dans ce projet, on n'a pas centré le travail sur les différentes architectures possibles que l'on pourrait utiliser pour les PINNs. On a en général utilisé des MLP (Multi Layer Perceptrons) en jouant un peu sur les activations, la profondeur ainsi que la largeur du réseau. En effet, la taille du réseau est d'importance non négligeable car elle peut définir son potentiel d'approximation. De plus, la profondeur joue un rôle important car ce sont les dernières couches qui vont s'occuper de la capturation de la fréquence de l'onde. Une idée serait de voir quels types d'architecture permettrait

d'obtenir des résultats plus satisfaisants, des durées d'entraînement plus courtes.

- **Activations:** Le choix de l'activation ou des activations s'avère être très important pour réussir à capturer la solution de l'EDP. Toujours dans le cas du guide d'onde, le meilleur choix est la tangente hyoerbolique (Tanh) qui est adapté à ce contexte de solution ondulatoire basée sur des sinusoïdes. D'autres choix comme ReLU, CELU, SoftSign ont fait que le réseau capture la solution nulle ou bien une mauvaise prédiction.

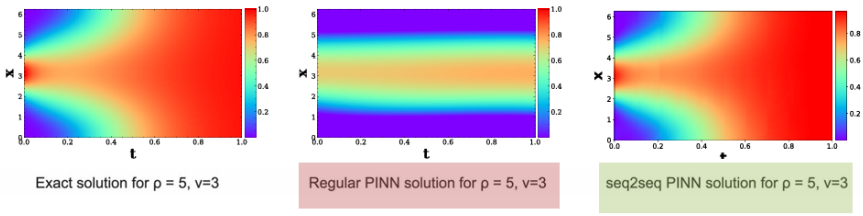


(a) Avec Softsign (b) Avec CeLU

- **Initialisation du réseau:** L'initialisation du réseau peut également influencer la prédiction de la solution. En effet, une initialisation nulle dans une stratégie qui priorise la loss résiduelle et dans un cas de source nulle, peut facilement conduire à la prédiction de la solution nulle. Une initialisation connue possible évitant ce problème s'appelle Xavier: tirer des échantillons d'une distribution normale tronquée centrée sur 0 avec $stddev = \sqrt{(2/(fan_{in} + fan_{out}))}$ où fan_{in} est le nombre d'unités d'entrée dans le tenseur de poids et fan_{out} est le nombre d'unités de sortie dans le tenseur de poids.
- **Losses:** Dans ce travail, nous ne nous sommes pas intéressés à l'étude de l'effet du choix des natures des loss (L1, MSE, SE, LASSO, Ridge..) ayant toujours opté pour la MSE. Un travail qui se veut être plus exhaustif pourrait s'intéresser à ce point.
- **Mémoire:** Une difficulté apparue lorsqu'on a voulu augmenté le nombre de points de collocation est la RAM limitée dont on dispose. Une idée possible est d'opter pour un entraînement en plusieurs batches. Cependant, un meilleur résultat n'est pas garanti sachant notamment les recommandations dans la littérature d'opter pour des entraînements full-batch pour les PINNs car les résolutions ne semblent pas marcher autrement.
- **Composante temporelle** Dans le cas d'une évolution dans le temps également (non traitée dans ce projet), la tâche devient nettement plus difficile et une approche classique de résolution du problème entier en une étape ne donne souvent pas une bonne prédiction. Une idée dans ce cas serait de faire un apprentissage séquentiel (seq2seq) en partageant le domaine temporelle en plusieurs petites tranches [2].

5 Conclusion

L'apprentissage de fonctions de hautes fréquences est important pour l'utilisation des PINNs dans des cas réels. Dans [4] et [5], il est montré que les réseaux de neurones apprennent moins



bien les fonctions de hautes fréquences. Les expérimentations menées dans la section 3.2 tendent à corroborer ce résultat. Différentes méthodes ont été mises en place pour remédier à ce biais spectral, comme l'apprentissage des fonctions de hautes fréquences par "cheminement", évoqué dans la section 3.2.
L'équilibrage des termes de la fonction de perte est un aspect crucial de l'entraînement des PINNs. L'équilibrage statique des pertes offre une simplicité mais nécessite un ajustement minutieux des poids, tandis que l'équilibrage dynamique des pertes adapte les poids pendant l'entraînement, offrant potentiellement une meilleure stabilité et précision. Les deux stratégies ont leurs mérites et peuvent être choisies en fonction du problème et des ressources de calcul disponibles. Dans nos travaux, nous avons souvent opté pour un équilibrage de loss statique ou un équilibrage dynamique avec un plateau de valeurs prédéterminées pour le coefficient résiduel.

References

[1] Rafael Bischof and Michael Kraus. Multi-objective loss balancing for physics-informed deep learning. 2021.

[2] Amir Gholami. Rethinking physics informed neural networks, https://www.youtube.com/watch?v=qymkuxh7tcy.

[3] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *Journal of Machine Learning Research*, 19(25):1–24, 2018.

[4] Basri Ronen, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. *Advances in Neural Information Processing Systems*, 32, 2019.

[5] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.