École nationale des ponts et chaussées

Institut Polytechnique de Paris

Département d'Ingénierie mathématique et informatique

Projet de département

Réseaux de neurones physiquement informés

Alexandre Gachenot Kyriann Loue Alexis Lucas Clément Mazzocchi
Mathis Metz Paul Saurin

Sous la supervision de
Antoine Bensalah Sofiane Haddad Ali Hebbal
Airbus Central Research & Technology

2025

Remerciements

Nous tenons à remercier l'École nationale des ponts et chaussées — Institut Polytechnique de Paris, et plus précisément le département d'Ingénierie mathématique et informatique, de nous permettre de travailler sur des thématiques intéressantes et actuelles, faisant l'objet d'une recherche scientifique abondante. Nous sommes reconnaissants de l'opportunité qui nous est faite de travailler sur ces sujets avec des chercheurs à l'interface entre les mondes académique et industriel. Nous remercions plus particulièrement Eric Duceau, président du département, David Gontier, responsable académique et Sandrine Guillerm, inspectrice des études.

Nous souhaitons également remercier les enseignants qui nous ont accompagné tout au long de l'année et notamment Xavier Clerc, responsable du module Techniques de développement logiciel dans le cadre duquel nous avons commencé nos réflexions sur le projet, et Maxence Lab.

Enfin, nous remercions évidemment Antoine Bensalah, Sofiane Haddad et Ali Hebbal qui nous ont transmis leurs connaissances et leurs conseils et avec lesquels nous avons eu plaisir à collaborer pendant plusieurs mois.

Sommaire

1 Introduction

Les équations aux dérivées partielles sont omniprésentes en physique : équation de Fourier en thermique, équation de Poisson en électrostatique, équation de d'Alembert en électromagnétisme, équation de Stokes en mécanique des fluides, équation de Schrödinger en quantique... Elles permettent de lier les évolutions spatio-temporelles de la fonction décrivant l'état du système aux termes sources, apportant une description analytique du phénomène physique considéré.

Soit un ouvert régulier $\Omega \subset \mathbb{R}^d$. Une équation aux dérivées partielles (du premier ordre en temps) s'écrit dans le cas général sous la forme suivante :

$$\begin{cases} \dfrac{\partial u}{\partial t}(\mathbf{x},t) + \mathcal{L}u(\mathbf{x},t) = f(\mathbf{x},t) & \forall \mathbf{x} \in \Omega, \forall t \in (0,T] \\ \mathcal{B}u(\mathbf{x},t) = g(\mathbf{x},t) & \forall \mathbf{x} \in \partial\Omega, \forall t \in [0,T] \\ u(\mathbf{x},0) = u_0(\mathbf{x}) & \forall \mathbf{x} \in \Omega \end{cases}$$

où u est la fonction décrivant l'état du système en tout point $\mathbf{x}$ de l'espace et à tout instant t. $\mathcal{L}$ et $\mathcal{B}$ sont des opérateurs différentiels agissant sur les variables d'espace. f et g sont les fonctions d'excitation du système. La première ligne décrit la physique à l'intérieur du domaine. Les deux suivantes donnent les conditions de bord (comportement de u à la frontière du domaine) et les conditions initiales.

Il est rare de pouvoir résoudre analytiquement une équation aux dérivées partielles. En effet, dès que les termes sources sont non-triviaux ou si les opérateurs différentiels sont trop sophistiqués, il faut faire appel à différentes méthodes pour déterminer u.

Méthodes classiques Les méthodes dites classiques consistent à utiliser des résultats d'analyse pour résoudre le problème. Il s'agit de chercher une solution dans un espace fonctionnel bien choisi. Une discrétisation du problème conduit à des méthodes de type éléments finis qui permettent de résoudre l'équation dans des domaines restreints. Il faut ensuite raccorder les solutions entre les différentes cellules pour obtenir une fonction de régularité voulue. Il est très souvent nécessaire de faire appel à une résolution numérique.

Ces méthodes sont très nombreuses et diversifiée et nous ne les aborderons pas dans ce document. Il est important de garder en tête qu'elles visent à résoudre l'équation physique sous-jacente au phénomène et d'en déduire la solution.

Méthodes d'apprentissage Les méthodes d'apprentissage se sont fortement développées depuis plusieurs décennies. Elles adoptent un paradigme radicalement différent de celui des méthodes classiques : au lieu de chercher à résoudre l'équation, c'est-à-dire comprendre le phénomène physique déterminant l'état du système, leur but est seulement de donner une solution acceptable sous forme de boîte noire en minimisant une fonction résiduelle prédéfinie.

On ne fournit donc pas l'équation régissant l'évolution spatio-temporelle du système mais des données, issues de mesures *in situ* ou de simulations numériques. Puis on mesure l'écart d'une première approximation de la solution, souvent éloignée de la réalité, avec la solution réelle. Le modèle possède un ensemble de paramètres qu'il peut ajuster pour se rapprocher de la solution.

Concrètement, le modèle évalue à chaque itération une fonction coût qui mesure l'écart entre l'approximation évaluée par le modèle et la solution réelle :

$$\mathcal{C}(\hat{u}_k) = \|\hat{u}_k - u\|$$

où $\hat{u}_k$ est l'approximation de la solution à l'itération k. De multiples choix de normes sont possibles pour évaluer cette fonction coût, mais dans tous les cas il est nécessaire d'échantilloner le domaine. On construit donc une discrétisation Ω_d du domaine Ω avec $|\Omega_d| = N_\Omega$. On doit aussi discrétiser l'intervalle $[0,T]$: on notera T_d cette discrétisation, avec $|T_d| = N_T$. Par exemple avec l'erreur quadratique moyenne, on a :

$$\mathcal{C}(\hat{u}_k) = \frac{1}{N_\Omega N_T} \sum_{\mathbf{x}_i \in \Omega_d} \sum_{t_j \in T_d} |\hat{u}_k(\mathbf{x}_i, t_j) - u(\mathbf{x}_i, t_j)|^2.$$

Réseaux de neurones physiquement informés Les réseaux de neurones physiquement informés (*physical informed neural networks*, PINNs en anglais) présentent une approche hybride entre les méthodes classiques et les méthodes d'apprentissage. En effet, ils consistent comme tous les réseaux de neurones à minimiser une fonction coût. Toutefois, ils ont la particularité d'intégrer la physique sous-jacente dans la fonction de coût.

Il s'agit d'une méthode récente : ils ont été introduits pour la première fois en 2019 dans [5].

Le modèle ne cherche plus à minimiser l'écart entre une approximation de la solution et la solution réelle, mais à minimiser l'écart de la solution approximée à la loi physique dont elle découle. Ainsi, la fonction de coût d'un PINN peut prendre la forme générale suivante :

$$\mathcal{C}(\hat{u}_k) = \underbrace{\left\| \frac{\partial \hat{u}_k}{\partial t} + \mathcal{L}\hat{u}_k - f \right\|}_{\mathcal{C}_\Omega} + \underbrace{\|\mathcal{B}\hat{u}_k - g\|}_{\mathcal{C}_{\partial\Omega}} + \underbrace{\|\hat{u}_k(\cdot, 0) - u_0\|}_{\mathcal{C}_0}.$$

Comme précédemment, on échantillonne le domaine en temps et en espace : on considère $\Omega_d \subset \Omega$, $\partial\Omega_d \subset \partial\Omega$ et $T_d \subset (0, T]$, de cardinaux respectifs $|\Omega_d| = N_\Omega$, $|\partial\Omega_d| = N_{\partial\Omega}$ et $|T_d| = N_T$. Cela donne par exemple avec l'erreur quadratique moyenne :

$$\begin{aligned}\mathcal{C}(\hat{u}_k) = &\frac{1}{N_\Omega N_T} \sum_{\mathbf{x}_i \in \Omega_d} \sum_{t_j \in T_d} \left| \frac{\partial \hat{u}_k}{\partial t}(\mathbf{x}_i, t_j) + \mathcal{L}\hat{u}_k(\mathbf{x}_i, t_j) - f(\mathbf{x}_i, t_j) \right|^2 \\ &+ \frac{1}{N_{\partial\Omega} N_T} \sum_{\mathbf{x}_i \in \partial\Omega_d} \sum_{t_j \in T_d} |\mathcal{B}\hat{u}_k(\mathbf{x}_i, t_j) - g(\mathbf{x}_i, t_j)|^2 \\ &+ \frac{1}{N_\Omega} \sum_{\mathbf{x}_i \in \Omega_d} |\hat{u}_k(\mathbf{x}_i, 0) - u_0(\mathbf{x}_i)|^2\end{aligned}$$

Le premier terme garantit que la loi physique sous-jacente soit respectée par la solution du modèle. Le second terme est un terme de bord, qui pénalise une solution ne respectant pas les conditions de bord prescrites. Le troisième terme teste la compatibilité de la solution du modèle avec les conditions initiales.

Notons que les dérivées apparaissant dans la fonction coût ne sont pas approximées numériquement (par différences finies par exemple), mais calculées via la différentiation automatique à partir de la structure du réseau de neurones. La différentiation automatique repose sur l'utilisation d'un graphe de calcul permettant de calculer efficacement les dérivées des quantités d'intérêt dans l'algorithme de rétropropagation. Nous montrons en annexe A comment calculer analytiquement ces dérivées d'intérêt. Toute la difficulté vient du fait que la fonction de coût fait intervenir les dérivées spatiales (et temporelles) de la solution. Il est de plus nécessaire de dériver par-rapport aux paramètres du réseau de neurones, ce qui conduit à des calculs complexes.

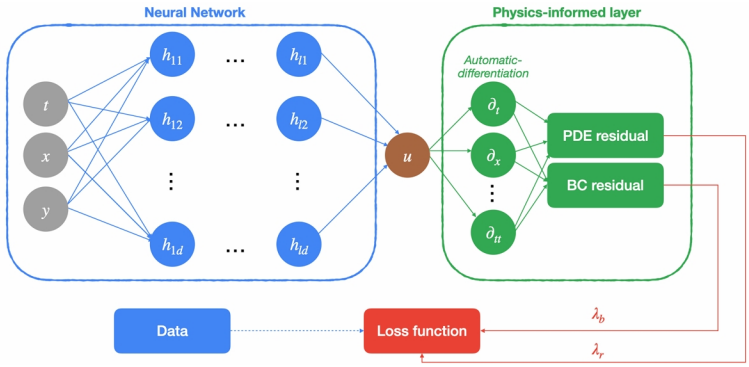


Figure 1: Architecture générale d'un PINN
(Crédit : Ali Hebbal)

Le besoin de solutions rapides et précises à des problèmes régis par des équations aux dérivées partielles est important, tant dans le milieu académique qu'industriel. Dans un contexte de très fort développement des techniques de machine learning, les PINNs représentent une alternative intéressante qui n'utilise que très peu de données. Depuis la publication de l'article fondateur [5] en 2018, une littérature abondante a exploré les multiples applications et prolongements des PINNs : amélioration de la robustesse aux discontinuités, correction des biais d'approximation, développement de schémas hybrides avec des méthodes numériques classiques, etc. Les PINNs et techniques dérivées bénéficient d'une popularité grandissante dans l'industrie en raison de leurs très bons résultats. Toutefois, leur caractère récent ne permet pas de déterminer précisément les causes de ce succès. Dans ce contexte, notre objectif est d'implémenter des PINNs sur quelques cas tests (équation de Helmholtz, tourbillon isentropique) afin d'évaluer leur capacité à approximer des solutions physiques sans données. Dans notre démarche, nous explorons la sensibilité des PINNs au choix des paramètres, aux conditions initiales et de bord, aux géométries utilisées, etc. Enfin, nous étudions quelques prolongements des PINNs : stratégies d'équilibrage de la fonction de coût, réseaux récurrents, PINNs dans les géométries complexes.

2 Modélisation et résolution

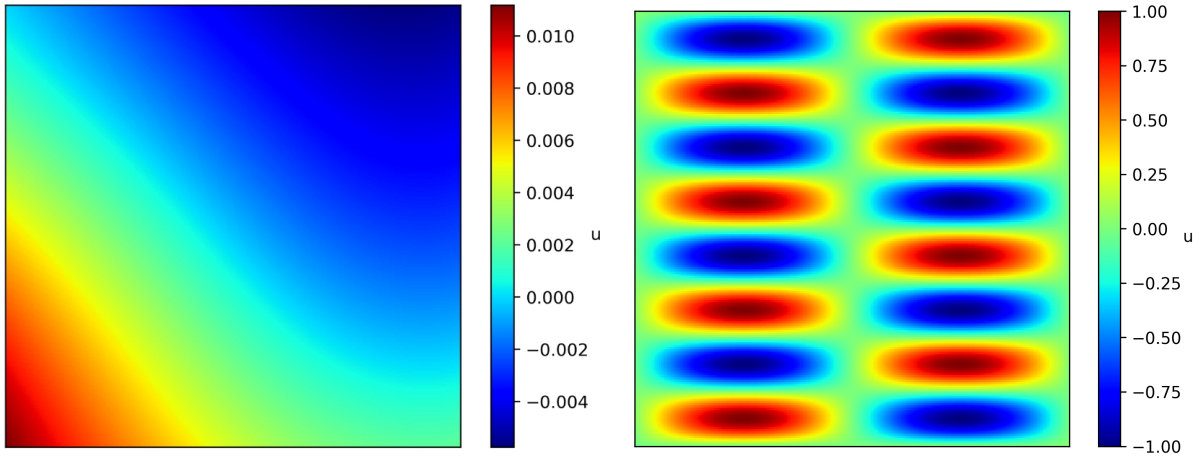
2.1 Exemple introductif : équation de Helmholtz

Nous avons d'abord utilisé un PINN pour la résolution de l'équation de Helmholtz en deux dimensions. L'équation de Helmholtz provient de la transformée de Fourier de l'équation des ondes classique $\frac{\partial^2 u}{\partial t^2} - c^2 \Delta u = -f$. En effet, en supposant une dépendance temporelle harmonique de fréquence ω, on obtient l'équation de Helmholtz. Elle décrit la propagation d'une onde dans l'espace en présence d'un éventuel terme source f.

Soit Ω un ouvert régulier de $\mathbb{R}^d$. L'équation de Helmholtz se formule ainsi :

$$\Delta u(\mathbf{x}) + k^2 u(\mathbf{x}) = -f(\mathbf{x}) \quad \forall x \in \Omega$$

$k = \frac{\omega}{c}$ est le nombre d'onde. Nous avons mené la résolution avec des conditions de Dirichlet mais la méthode est aisément transposable à d'autres conditions de bord.



(a) $k = 1$, $f = 0$ dans Ω, $u(\mathbf{x}) = \cos\left(\frac{x+y}{2}\right) \forall \mathbf{x} \in \partial\Omega$ (b) $k = 1$, $f(\mathbf{x}) = \sin(2\pi x)\sin(6\pi y) \forall \mathbf{x} \in \Omega$, $u = 0$ sur $\partial\Omega$

Figure 2: Deux exemples de résolution de l'équation de Helmholtz par un PINN

2.2 Tourbillon isentropique en deux dimensions

Le second problème sur lequel nous avons travaillé est le tourbillon isentropique. Nous considérons un tourbillon local se déplaçant rectilignement dans un fluide régi par les équations d'Euler. Notre objectif est de réussir à capturer la dynamique du tourbillon, sans perdre en exactitude dans sa description locale. En d'autres termes, le PINN doit être capable de modéliser le déplacement du fluide tout en assurant une bonne description du caractère tourbillonnaire.

De plus, le PINN doit être capable d'extrapoler la solution sur un intervalle temporel sur lequel il n'a pas été entraîné.

Nous travaillons ici en deux dimensions spatiales et utilisons deux types de PINNs : le premier est un *fully connected*, dont nous savons qu'il décrit bien les phénomènes stationnaires mais pourrait échouer à capturer la dynamique ; nous comparons donc nos résultats avec un PINN hybride, qui couple un *fully connected* utilisé dans l'horizon temporel d'entrainement, et un réseau de neurones récurrent utilisé lors de l'extrapolation.

2.2.1 Cadre physique : équations d'Euler

En mécanique des fluides, on décrit le fluide considéré à l'aide d'une série d'équations correspondant aux moments de la fonction de distribution des vitesses. Ces équations prennent la forme de lois de conservation, dont les premières sont données en annexe B.1.

Fonction de courant On définit la fonction de courant du tourbillon régularisé centré en (x_c, y_c) par

$$\psi(x,y) = \frac{b}{2\pi}\exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right)$$

avec $r = \sqrt{(x-x_c)^2 + (y-y_c)^2}$, et R la taille caractéristique du phénomène. Cette fonction décrit un tourbillon régularisé dont l'intensité est contrôlée par le paramètre b.

Champs de vitesse Les perturbation aux champs de vitesse induites par le tourbillon s'obtiennent à partir de la fonction de courant via

$$u'_x = \frac{\partial \psi}{\partial y}, \quad u'_y = -\frac{\partial \psi}{\partial x}.$$

Les calculs permettant d'obtenir les résultats ci-dessous sont donnés en annexe B.2.

$$\boxed{u_x = u_{x,\infty} - \frac{b}{2\pi}\frac{y-y_c}{R}\exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right)}$$

$$\boxed{u_y = u_{y,\infty} + \frac{b}{2\pi}\frac{x-x_c}{R}\exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right)}$$

On peut aussi obtenir les champs de vitesse en coordonnées cylindriques avec les formules

$$u'_r = \frac{1}{r}\frac{\partial \psi}{\partial \theta}, \quad u'_\theta = -\frac{\partial \psi}{\partial r}.$$

On va ici les déduire des champs en coordonnées cartésiennes. En effet, en se plaçant dans le référentiel du centre du tourbillon, on a $u'_r = \dfrac{u'_x}{\cos(\theta)}$ et $u'_\theta = \dfrac{u'_y}{\sin(\theta)}$ avec $\cos(\theta) = \dfrac{x-x_c}{r}$ et $\sin(\theta) = \dfrac{y-y_c}{r}$. On en déduit :

$$\boxed{u'_r = 0}$$

et

$$\boxed{u'_\theta = \frac{b}{2\pi}\frac{r}{R}\exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right)}.$$

Champs de température et de densité En injectant les champs de vitesse dans les équations d'Euler, on déterminer les champs de température et de densité. La démarche est présentée en annexe B.3.

$$\boxed{T(r) = T_\infty - \frac{b^2}{8\pi^2 c_P} \exp\left(1 - \frac{r^2}{R^2}\right)}$$

$$\boxed{\rho(r) = \rho_\infty \left(1 - \frac{\rho_\infty}{p_\infty} \frac{b^2}{8\pi^2} \frac{\gamma - 1}{\gamma} \exp\left(1 - \frac{r^2}{R^2}\right)\right)^{\frac{1}{\gamma - 1}}}$$

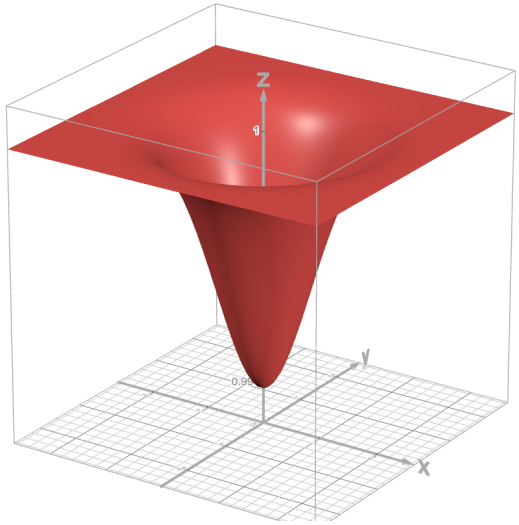


Figure 3: Profil de densité ($\rho_\infty = 1$, $p_\infty = 1$, $b = 0.5$, $\gamma = 1.4$, $R = 1$)

2.2.2 Résolution par un PINN et stratégies d'équilibrage de la fonction de coût

Nous avons résolu le problème précédent par un PINN *fully connected*. Nous effectuons la résolution sur l'intervalle de temps $t \in [0, 20]$. Puis, nous réalisons une inférence sur $[20, 40]$ pour évaluer la capacité du PINN à capturer la dynamique en-dehors de l'intervalle d'entraînement.

Un enjeu important pour l'entraînement des PINNs est la manière d'équilibrer les poids de la fonction de coût dans le domaine Ω et sur la frontière $\partial\Omega$. On introduit les notations suivantes :

$$\mathcal{C}(\hat{u}_k) = \lambda_\Omega \mathcal{C}_\Omega + \lambda_{\partial\Omega} \mathcal{C}_{\partial\Omega} + \lambda_0 \mathcal{C}_0$$

Le premier choix d'équilibrage des pertes consiste à fixer arbitrairement $\lambda_\Omega = \lambda_{\partial\Omega} = \lambda_0 = 1$. Cependant, ce choix est purement heuristique et il existe des méthodes plus pertinentes pour ajuster dynamiquement ces poids au cours de l'entraînement. En effet, il est possible d'utiliser des méthodes dynamiques où les coefficients sont mis à jour à chaque itération.

GradNorm La méthode GradNorm est introduite dans [2]. Elle propose une stratégie d'équilibrage des pertes de la forme $L = \sum_i w_i(t) L_i(t)$, où les poids $w_i(t)$ sont ajustés à chaque étape d'entraînement. L'objectif est d'assurer que toutes les tâches progressent à un rythme similaire, en pénalisant les gradients trop grands ou trop faibles lors de la rétropropagation.

On introduit les quantités suivantes :

- $G_W^{(i)}(t) = \|\nabla_W (w_i(t)L_i(t))\|_2$: norme L^2 du gradient de la perte pondérée par $w_i(t)$ pour la tâche i (calculée sur les poids W partagés),
- $\overline{G}(t) = \frac{1}{N}\sum_{i=1}^{N} G_W^{(i)}(t)$: moyenne des normes de gradients sur toutes les tâches,
- $\tilde{L}_i(t) = \frac{L_i(t)}{L_i(0)}$: perte relative normalisée, permettant de suivre le taux d'entraînement de la tâche i,
- $r_i(t) = \frac{\tilde{L}_i(t)}{\frac{1}{N}\sum_{j=1}^{N}\tilde{L}_j(t)}$: taux d'entraînement inverse relatif de la tâche i.

Pour chaque tâche i, on souhaite que la norme de gradient $G^{(i)}(t)$ atteigne la valeur cible suivante :

$$G^{(i)}(t) \to \overline{G}(t) \cdot [r_i(t)]^{\alpha}$$

où $\alpha > 0$ est un hyperparamètre contrôlant la force du rééquilibrage entre les tâches.

On définit alors la fonction de perte GradNorm :

$$\mathcal{L}_{\text{grad}}(t; w_i(t)) = \sum_i \left\| G^{(i)}(t) - \overline{G}(t) \cdot [r_i(t)]^{\alpha} \right\|_1$$

Cette fonction pénalise les écarts entre les normes de gradients actuelles et les normes cibles. Lors du calcul du gradient de $\mathcal{L}_{\text{grad}}$, les termes cibles $\overline{G}(t) \cdot [r_i(t)]^{\alpha}$ sont considérés comme constants pour éviter que les poids $w_i(t)$ ne dérivent indûment.

L'algorithme complet est donné en annexe C.

SoftAdapt SoftAdapt est un autre algorithme d'agrégation de pertes. Il est notamment proposé dans [3] et introduit une méthode d'adaptation dynamique des poids dans une fonction de perte composée.

On considère une fonction de perte de la forme :

$$L = \sum_{k=1}^{n} w_k^i L_k^i$$

où les poids w_k^i sont mis à jour à chaque itération i afin de refléter l'évolution des sous-pertes L_k^i. L'idée de SoftAdapt est de pondérer les composantes en fonction de leur performance récente.

Pour cela, on définit :

$$s_k^i = L_k(x_i) - L_k(x_{i-1})$$

comme l'évolution de la perte k entre deux itérations successives (autrement dit, une approximation de sa dérivée temporelle). Ensuite, on met à jour les poids selon la formule suivante (variante *loss-weighted*) :

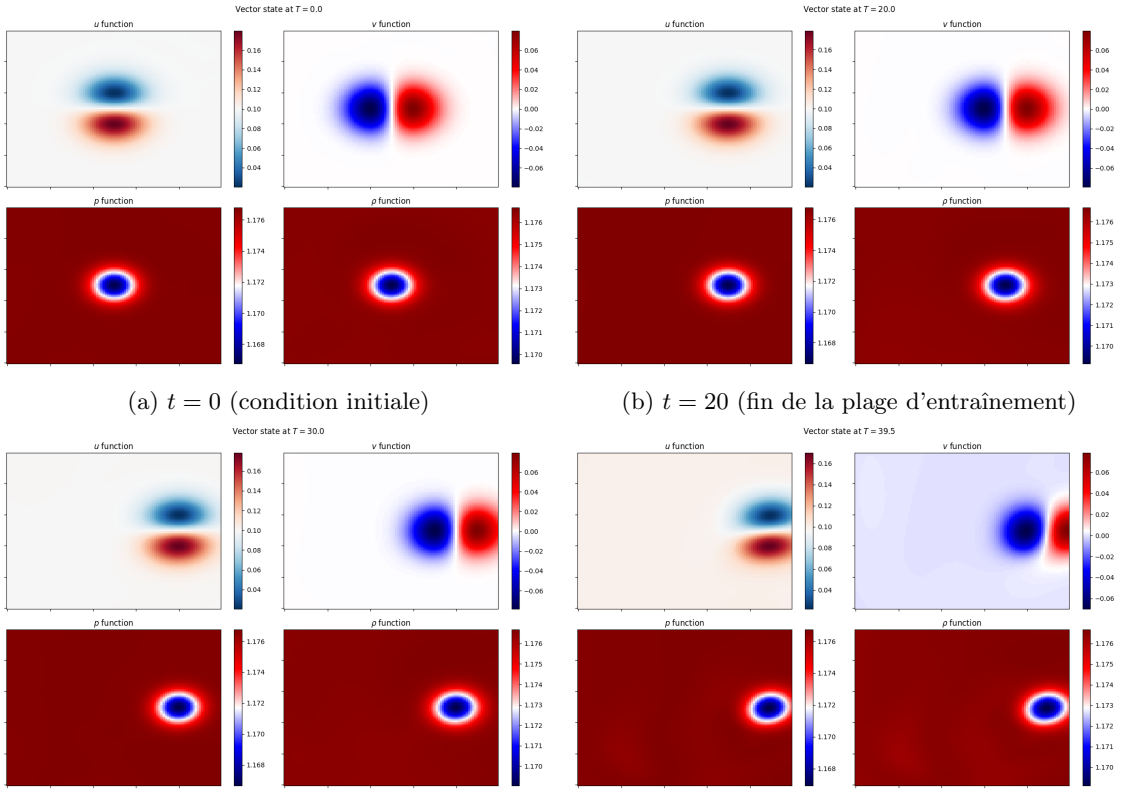
$$w_k^i = \frac{L_k^i \, e^{\beta s_k^i}}{\sum_{\ell=1}^{n} L_\ell^i \, e^{\beta s_\ell^i}}$$

Le paramètre $\beta \in \mathbb{R}$ permet de moduler la sensibilité aux changements de pertes :

- Si $\beta > 0$, on favorise les composantes dont la perte augmente (pire performance),
- Si $\beta < 0$, on favorise celles dont la perte diminue (meilleure performance),
- Si $\beta = 0$, toutes les composantes sont pondérées de manière égale.

Cette formule permet d'adapter dynamiquement les poids des différentes tâches en fonction de leur comportement récent, ce qui peut aider à éviter qu'une tâche domine l'entraînement ou soit négligée.

2.2.3 Résultats



(a) $t = 0$ (condition initiale)

(b) $t = 20$ (fin de la plage d'entraînement)

(c) $t = 30$ (inférence)

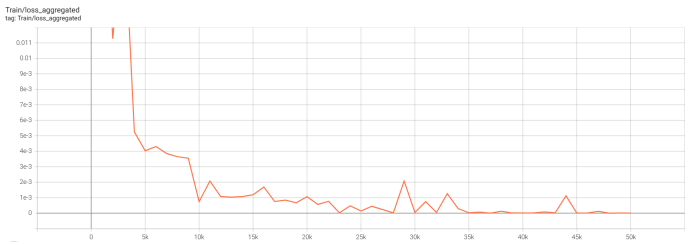
(d) $t = 39,5$ (inférence)

Figure 4: Résolution et inférence du tourbillon avec un PINN

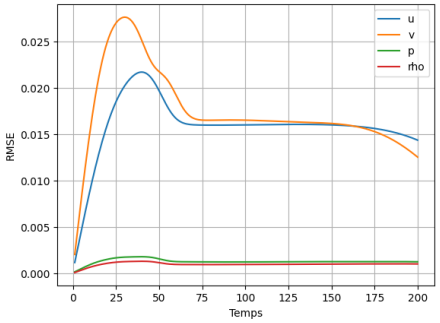
Sur la figure sont représentés les champs de vitesses (selon x et y), de température et de pression à différents instants t. Les deux premiers clichés ($t = 0$ et $t = 20$) montrent des solutions pour des pas de temps sur lesquels le PINN a été entraîné. On constate que les solutions sont cohérentes qualitativement : la forme du tourbillon est respectée et on constate un déplacement vers les x croissants.

Les deux clichés suivants montrent des solutions inférées par le PINN sur des pas de temps hors de sa plage d'entraînement : ayant capturé la dynamique du tourbillon sur les vingt premières secondes, le PINN parvient ensuite à extrapoler le mouvement sur les vingt secondes suivantes. On constate que les solutions sont toujours qualitativement satisfaisantes, mais que à $t = 39,5$, les champs de vitesse commencent à se déformer : on atteint les limites d'inférence du PINN.

Nous comparons ensuite l'efficacité de l'entraînement pour différentes fonctions de coût : GradNorm, SoftAdapt et une fonction de coût customisée. Cette dernière suit le même principe de fonctionnement que la fonction de coût GradNorm mais réduit l'importance des conditions initiales à chaque itération. Nous présentons ci-après l'évolution des fonctions de coût au cours de l'entraînement et l'erreur quadratique moyenne de la solution approximée par le PINN par-rapport à la solution analytique calculée en section 2.2.1.



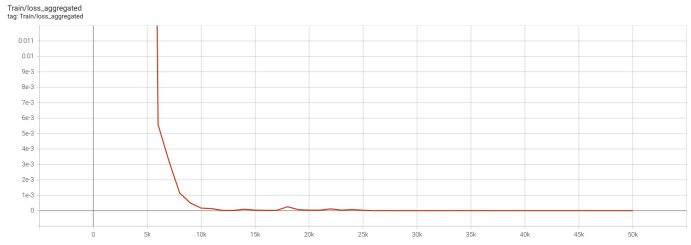
(a) Résidu



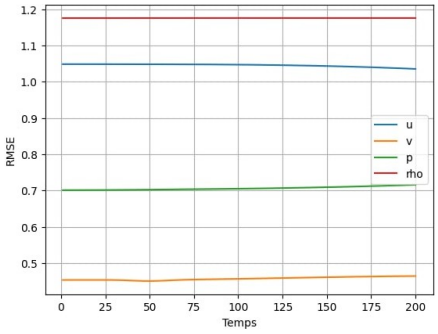
(b) Erreur quadratique moyenne

Figure 5: Fonction de coût non-équilibrée

Avec une fonction de coût sans stratégie d'équilibrage ($\lambda_\Omega = \lambda_{\partial\Omega} = \lambda_0$), les résultats obtenus sont assez satisfaisants, avec une erreur quadratique moyenne faible pour la plupart des composantes. Les résultats sont légèrement moins bons pour le champ de pression. Des stratégies d'équilibrage sont envisageables pour améliorer les résultats.



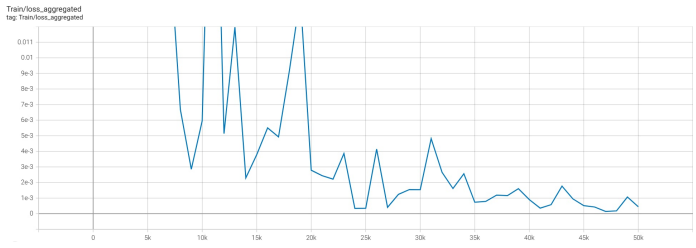
(a) Résidu



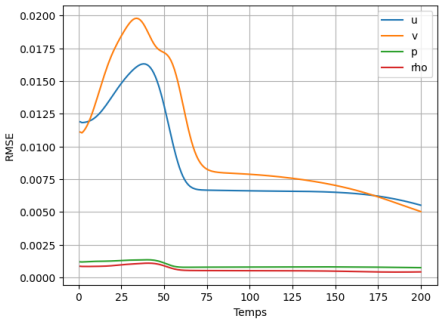
(b) Erreur quadratique moyenne

Figure 6: Fonction de coût GradNorm

Les résultats obtenus avec la fonction de coût GradNorm ne sont pas satisfaisants dans le cas du tourbillon. En effet, l'erreur quadratique moyenne reste élevée ce qui signifie que la solution approximée par le PINN diffère de la solution analytique.

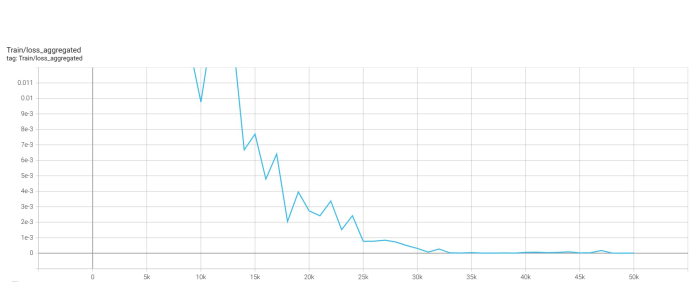


(a) Résidu

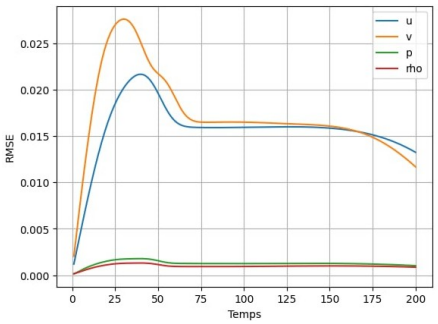


(b) Erreur quadratique moyenne

Figure 7: Fonction de coût SoftAdapt



(a) Résidu



(b) Erreur quadratique moyenne

Figure 8: Fonction de coût customisée

On obtient des résultats bien meilleurs avec la fonction de coût SoftAdapt et la fonction de coût customisée. En effet, les erreurs quadratiques moyennes sont bien plus faibles. Les résultats sont particulièrement satisfaisants pour les champs de pression et de densité, avec des erreurs constantes très faibles. Pour les champs de vitesse, on constate une forme caractéristique de l'erreur quadratique moyenne en fonction du temps, que l'on retrouve avec divers jeux de paramètres.

2.3 Couplage avec un réseau de neurones récurrent

Étant donné que le problème considéré est instationnaire, il est possible qu'un réseau *fully connected* ne parvienne pas à capturer précisément la dynamique du tourbillon en-dehors de la plage d'entraînement. Les réseaux récurrents sont eux, bien plus adaptés à ce type de problèmes. Nous avons donc tenté une deuxième approche, qui couple un PINN *fully connected* avec un réseau de neurones récurrent.

Plus précisément, nous entraînons le PINN sur la plage de temps $t \in [0, 20]$. Puis, nous utilisons le PINN pour entraîner un réseau de neurones récurrent Seq2seq qui, en connaissance de la solution sur l'intervalle $[t_1, t_2]$, peut prédire la solution sur l'intervalle $[t_1 + \delta t, t_2 + \delta t]$. Enfin, nous évaluons les performances du réseau de neurones récurrent sur l'intervalle $[20, 40]$ (qui ne fait pas partie de sa plage d'entraînement.

Les résultats obtenus avec cette approche hybride ne sont pas significativement meilleurs que ceux obtenus précédemment. Le comportement de u et v reste similaire, tant dans cette erreur quadratique moyenne que dans les précédentes. En revanche, on observe de manière inattendue un comportement anormal pour ρ et p, en désaccord avec les résultats antérieurs. Ce phénomène pourrait s'expliquer par un mauvais paramétrage du réseau de neurones, une implémentation incorrecte ou un entraînement insuffisant. Il est également possible que le fonctionnement du réseau récurrent soit moins adapté à l'évolution des variables ρ et p, contrairement à un simple PINN.

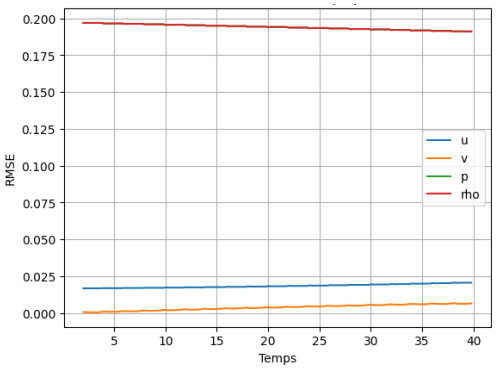


Figure 9: Erreur quadratique moyenne pour le réseau de neurones récurrent

2.4 Software et hardware

Les PINNs sont un outil récent (mentionnés pour la première fois dans [5] en 2018) évoluant très rapidement. La question de leur implémentation a été centrale dans ce projet.

La question s'est rapidement posée de choisir une implémentation complète, en s'appuyant sur les briques fondamentales de PyTorch et Keyras, ou d'utiliser une librairie. L'avantage des librairies est de proposer des architectures déjà implémentées et directement utilisables. Leur inconvénient est qu'elles peuvent être plus lourdes à utiliser.

Nous avons opté pour l'utilisation de la librairie PhysicsNeMo de NVIDIA, qui propose de nombreuses architectures de PINNs et qui laisse une grande marge de manœuvre grâce aux nombreux paramètres ajustables. L'utilisation de PhysicsNeMo suppose l'installation locale d'une machine Linux ou d'une machine virtuelle Linux et l'utilisation d'un conteneur via Docker. Il faut ensuite travailler en environnement fermé et simulé pour avoir accès aux fonctionnalités d'un IDE pour le développement. Il est nécessaire de posséder une carte graphique NVIDIA avec suffisamment de VRAM.

Ainsi, l'utilisation de PhysicsNeMo pose de nombreux défis techniques :

- L'utilisation locale de la librairie n'est possible que sur les machines équipées d'un cœur GPU.
- L'utilisation sur des plateformes *cloud* telles que Google Colab est lourde et peu adaptée : incompatibilité de versions Python, Colab et PhysicsNeMo, temps d'installation long se renouvelant à chaque session de travail, etc.
- Les limites d'utilisation des plateformes *cloud* (temps d'utilisation et mémoire RAM) limitent l'importance des entraînements réalisables.

Enfin, le caractère récent des PINNs et de PhysicsNeMo a pour conséquence une documentation technique lacunaire.

3 Conclusions et perspectives

Les PINNs sont un sujet de recherche très actif et en constant développement. Nous n'avons évoqué ici que quelques applications, mais de multiples prolongements sont possibles : PINNs paramétriques, PINNs variationnels, etc.

Comme nous l'avons mentionné dans l'introduction, l'avantage des PINNs devant les méthodes de résolution classiques (schémas d'éléments finis) peut résider dans leur capacité à produire des solutions satisfaisantes dans des géométries complexes, que les schémas numériques intègrent mal. Nous proposons donc un apercu des PINNs pour les géométries complexes sur lesquels nous avons travaillé, bien que nous ne les ayons pas implémentés en pratique. Une description précise des techniques employées pour l'utilisation des PINNs dans les géométries complexes est présentée en annexe D.

Les résultats les plus convaincants ont été obtenus sur le cas du tourbillon isentropique en deux dimensions. Le réseau *fully connected* s'est révélé capable de capturer fidèlement la structure tourbillonnaire dans l'intervalle d'entraînement, et d'en extrapoler qualitativement la dynamique sur un intervalle temporel ultérieur. Ce comportement met en évidence une capacité de généralisation intéressante des PINNs, même sans supervision par données.

L'implémentation de stratégies d'équilibrage dynamique de la fonction de coût, comme GradNorm ou SoftAdapt, a constitué un apport significatif pour la stabilité de l'entraînement. Ces approches, empruntées à l'apprentissage multi-tâches, se sont révélées efficaces pour éviter qu'un terme de perte ne domine les autres, améliorant ainsi la convergence.

Nous avons en revanche rencontré des difficultés dans l'approche hybride couplant PINN et réseau récurrent. Bien que prometteuse sur le papier, cette méthode n'a pas permis d'obtenir une solution stable : la structure tourbillonnaire s'estompe rapidement hors de l'intervalle d'apprentissage. Ce résultat suggère des limitations dans l'architecture ou l'entraînement du modèle récurrent.

Par ailleurs, nous avons exploré les méthodes d'imposition stricte des conditions de bord dans des géométries complexes, en modifiant l'ansatz du réseau. Cette approche offre des perspectives intéressantes pour l'extension des PINNs à des configurations industrielles.

Plusieurs limites restent à surmonter. L'usage de librairies spécialisées comme PhysicsNeMo a mis en lumière les contraintes pratiques liées à ces outils, notamment en termes de documentation et de compatibilité logicielle. Les difficultés rencontrées dans l'utilisation de certaines plateformes (comme Google Colab) restreignent les possibilités d'entraînement.

Les perspectives de ce travail sont nombreuses. Il serait pertinent de reprendre l'approche hybride avec des architectures récurrentes plus performantes, d'envisager l'extension à des cas tridimensionnels, ou encore d'explorer des couplages entre PINNs et méthodes numériques classiques. Enfin, l'étude de variantes telles que les PINNs variationnels ou paramétriques ouvre la voie à une meilleure flexibilité dans la modélisation.

Les PINNs apparaissent ainsi comme une voie prometteuse pour la simulation numérique guidée par la physique, particulièrement adaptée à des contextes où les données sont rares ou coûteuses. Ce projet aura permis de confirmer leur potentiel, tout en mettant en lumière les défis qu'il reste à relever pour les rendre pleinement opérationnels. Soulevons notamment l'absence de garanties de convergence et de bornes d'erreur théoriques pour les PINNs, contrairement aux méthodes classiques pour lesquelles on contrôle à la fois la vitesse de convergence et la validité de la solution. Obtenir des garanties théoriques sur le comportement des PINNs est une condition *sine qua non* de leur utilisation à grande échelle dans des projets industriels.

A Algorithmes de rétropropagation

Les calculs ci-après sont adaptés de [1].

A.1 Cas général

On considère un réseau de neurones profond *fully connected* de $L+1$ couches :

- entrées : couche $l = 0$;
- couches cachées : couches $0 < l < L$;
- sorties : couche $l = L$.

Ce réseau de neurones possède différents paramètres ajustables que nous noterons ainsi :

- le neurone j de la couche l a un biais b_j^l ;
- la transition entre le neurone k de la couche $l-1$ et le neurone j de la couche l fait intervenir un poids w_{jk}^l.
- la fonction d'activation de la couche l est notée σ_l.

Enfin, on notera $x_1, \ldots, x_N$ les entrées du réseau, z_j^l la sortie du neurone j de la couche l avant l'application de la fonction d'activation, et $y_j^l = \sigma_l(z_j^l)$. Avec ces notations, les sorties du réseau de neurones sont donc $y_1^L, \ldots, y_N^L$.

L'équation régissant la transition d'une couche $l-1$ à une couche l est

$$z_j^l = \sum_k w_{jk}^l \sigma_{l-1}(z_k^{l-1}) + b_j^l = \sum_k w_{jk}^l y_k^{l-1} + b_j^l$$

Le but d'un réseau de neurones classique est de minimiser une fonction de coût $\mathcal{C}(y, y^L)$ où y est la solution (analytique, numérique ou expérimentale) fournie est y^L est la sortie du réseau de neurones. Plus précisément, il s'agit d'identifier les paramètres du réseau de neurones (biais et poids) qui minimisent la fonction de coût :

$$w^*, b^* = \arg\min_{w,b} \mathcal{C}(y, y^L)$$

En effet, l'objectif du réseau de neurones n'est pas (seulement) de bien reproduire les données d'entrée, mais de pouvoir extrapoler sur des données inconnues.

Pour réaliser la minimisation, on a besoin de calculer les dérivées partielles de la fonction de coût par-rapport à chacun des paramètres du réseau :

$$\begin{cases} \dfrac{\partial \mathcal{C}}{\partial w_{jk}^l} = \dfrac{\partial \mathcal{C}}{\partial z_j^l} \dfrac{\partial z_j^l}{\partial w_{jk}^l} = \dfrac{\partial \mathcal{C}}{\partial z_j^l} y_k^{l-1} \\ \dfrac{\partial \mathcal{C}}{\partial b_j^l} = \dfrac{\partial \mathcal{C}}{\partial z_j^l} \dfrac{\partial z_j^l}{\partial b_j^l} = \dfrac{\partial \mathcal{C}}{\partial z_j^l} \end{cases}$$

Notre problème se ramène donc au calcul de $\delta_j^l := \dfrac{\partial \mathcal{C}}{\partial z_j^l}$. Le cœur de l'algorithme de rétropropagation est le calcul de cette quantité, par une récursion inversée :

$$\begin{cases} \delta_j^L = \dfrac{\partial \mathcal{C}}{\partial y_j^L} \dfrac{\partial y_j^L}{\partial z_j^L} = \dfrac{\partial \mathcal{C}}{\partial y_j^L} \sigma_L'(z_j^L) \\ \delta_j^l = \displaystyle\sum_k \dfrac{\partial \mathcal{C}}{\partial z_k^{l+1}} \dfrac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma_l'(z_j^l) \end{cases}$$

A.2 Cas des PINNs

Dans un PINN, la fonction de coût fait intervenir la solution de l'équation aux dérivées partielles et ses dérivées. On a donc besoin de calculer les dérivées de la sortie du réseau de neurones par-rapport aux coordonnées spatiales.

Dérivées premières

$$\begin{cases} \dfrac{\partial y_j^L}{\partial x_i} = \sigma_L'(z_j^L)\dfrac{\partial z_j^L}{\partial x_i} \\ \dfrac{\partial z_j^l}{\partial x_i} = \displaystyle\sum_k w_{jk}^l \sigma_{l-1}'(z_k^{l-1})\dfrac{\partial z_k^{l-1}}{\partial x_i} \end{cases}$$

Dérivées secondes

$$\begin{cases} \dfrac{\partial^2 y_j^L}{\partial x_i^2} = \sigma_L''(z_j^L)\left(\dfrac{\partial z_j^L}{\partial x_i}\right)^2 + \sigma_L'(z_j^L)\dfrac{\partial^2 z_j^L}{\partial x_i^2} \\ \dfrac{\partial^2 z_j^l}{\partial x_i^2} = \displaystyle\sum_k w_{jk}^l \left(\sigma_{l-1}''(z_k^{l-1})\left(\dfrac{\partial z_k^{l-1}}{\partial x_i}\right)^2 + \sigma_{l-1}'(z_k^{l-1})\dfrac{\partial^2 z_k^{l-1}}{\partial x_i^2}\right) \end{cases}$$

On a ensuite besoin de calculer les dérivées par-rapport aux paramètres du réseau des dérivées spatiales de la solution calculée. On peut reproduire la démarche décrite dans le cas général en choisissant comme fonction de coût $\mathcal{C}(y^L) = y^L$:

$$\begin{cases} \dfrac{\partial y_m^L}{\partial w_{jk}^l} = \dfrac{\partial y_m^L}{\partial z_j^l}\dfrac{\partial z_j^l}{\partial w_{jk}^l} = \dfrac{\partial y_m^L}{\partial z_j^l} y_k^{l-1} \\ \dfrac{\partial y_m^L}{\partial b_j^l} = \dfrac{\partial y_m^L}{\partial z_j^l}\dfrac{\partial z_j^l}{\partial b_j^l} = \dfrac{\partial y_m^L}{\partial z_j^l} \end{cases}$$

En notant $\delta_j^l := \dfrac{\partial y_m^L}{\partial z_j^l}$, on a

$$\begin{cases} \delta_m^L = \sigma_L'(z_m^L) \\ \delta_j^l = \displaystyle\sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma_l'(z_j^l) \end{cases}$$

Dérivées premières On souhaite calculer $\dfrac{\partial^2 y_m^L}{\partial x_i \partial w_{jk}^l}$ et $\dfrac{\partial^2 y_m^L}{\partial x_i \partial b_j^l}$:

$$\begin{cases} \dfrac{\partial^2 y_m^L}{\partial x_i \partial w_{jk}^l} = \dfrac{\partial}{\partial x_i}\left(\dfrac{\partial y_m^L}{\partial w_{jk}^l}\right) = \dfrac{\partial}{\partial x_i}\left(\delta_j^l y_k^{l-1}\right) = y_k^{l-1}\dfrac{\partial \delta_j^l}{\partial x_i} + \delta_j^l \dfrac{\partial y_k^{l-1}}{\partial x_i} = y_k^{l-1}\dfrac{\partial \delta_j^l}{\partial x_i} + \delta_j^l \sigma_{l-1}'(z_k^{l-1})\dfrac{\partial z_k^{l-1}}{\partial x_i} \\ \dfrac{\partial^2 y_m^L}{\partial x_i \partial b_j^l} = \dfrac{\partial}{\partial x_i}\left(\dfrac{\partial y_m^L}{\partial b_j^l}\right) = \dfrac{\partial \delta_j^l}{\partial x_i} \end{cases}$$

On se concentre donc sur le calcul de $\dfrac{\partial \delta_j^l}{\partial x_i}$:

$$\begin{cases} \dfrac{\partial \delta_m^L}{\partial x_i} = \sigma_L''(z_m^L)\dfrac{\partial z_m^L}{\partial x_i} \\ \dfrac{\partial \delta_j^l}{\partial x_i} = \displaystyle\sum_k w_{kj}^{l+1}\left(\dfrac{\partial \delta_k^{l+1}}{\partial x_i}\sigma_l'(z_j^l) + \delta_k^{l+1}\sigma_l''(z_j^l)\dfrac{\partial z_j^l}{\partial x_i}\right) \end{cases}$$

Dérivées secondes On souhaite calculer $\frac{\partial^3 y_m^L}{\partial x_i^2 \partial w_{jk}^l}$ et $\frac{\partial^3 y_m^L}{\partial x_i^2 \partial b_j^l}$:

$$\begin{aligned}
\frac{\partial^3 y_m^L}{\partial x_i^2 \partial w_{jk}^l} &= \frac{\partial}{\partial x_i}\left(\frac{\partial^2 y_m^L}{\partial x_i \partial w_{jk}^l}\right) = \frac{\partial}{\partial x_i}\left(y_k^{l-1}\frac{\partial \delta_j^l}{\partial x_i} + \delta_j^l \sigma'_{l-1}(z_k^{l-1})\frac{\partial z_k^{l-1}}{\partial x_i}\right) \\
&= \frac{\partial y_k^{l-1}}{\partial x_i}\frac{\partial \delta_j^l}{\partial x_i} + y_k^{l-1}\frac{\partial^2 \delta_j^l}{\partial x_i^2} + \frac{\partial \delta_j^l}{\partial x_i}\sigma'_{l-1}(z_k^{l-1})\frac{\partial z_k^{l-1}}{\partial x_i} + \delta_j^l \frac{\partial \sigma'_{l-1}(z_k^{l-1})}{\partial x_i}\frac{\partial z_k^{l-1}}{\partial x_i} + \delta_j^l \sigma'_{l-1}(z_k^{l-1})\frac{\partial^2 z_k^{l-1}}{\partial x_i^2} \\
&= \left(\sigma''_{l-1}(z_k^{l-1})\left(\frac{\partial z_k^{l-1}}{\partial x_i}\right)^2 + \sigma'_{l-1}(z_k^{l-1})\frac{\partial^2 z_k^{l-1}}{\partial x_i^2}\right)\delta_j^l + 2\sigma'_{l-1}(z_k^{l-1})\frac{\partial z_k^{l-1}}{\partial x_i}\frac{\partial \delta_j^l}{\partial x_i} + y_k^{l-1}\frac{\partial^2 \delta_j^l}{\partial x_i^2}
\end{aligned}$$

$$\frac{\partial^3 y_m^L}{\partial x_i^2 \partial b_j^l} = \frac{\partial^2}{\partial x_i^2}\left(\frac{\partial y_m^L}{\partial b_j^l}\right) = \frac{\partial^2 \delta_j^l}{\partial x_i^2}$$

On se concentre donc sur le calcul de $\frac{\partial^2 \delta_j^l}{\partial x_i^2}$:

$$\begin{cases}
\frac{\partial^2 \delta_m^L}{\partial x_i^2} = \sigma_L^{(3)}(z_m^L)\frac{\partial z_m^L}{\partial x_i} + \sigma''_L(z_m^L)\frac{\partial^2 z_m^L}{\partial x_i^2} \\
\frac{\partial^2 \delta_j^l}{\partial x_i^2} = \sum_k w_{kj}^{l+1}\left(\frac{\partial^2 \delta_k^{l+1}}{\partial x_i^2}\sigma'_l(z_j^l) + 2\frac{\partial \delta_k^{l+1}}{\partial x_i}\sigma''_l(z_j^l)\frac{\partial z_j^l}{\partial x_i} + \delta_k^{l+1}\left(\sigma_l^{(3)}(z_j^l)\left(\frac{\partial z_j^l}{\partial x_i}\right)^2 + \sigma''_l(z_j^l)\frac{\partial^2 z_j^l}{\partial x_i^2}\right)\right)
\end{cases}$$

B Équations d'Euler

Les dérivations détaillées ci-après s'appuient sur les résultats exposés dans [7].

B.1 Lois de conservation

Soit une quantité intensive ϕ, entraînée à la vitesse $\mathbf{u}$ avec un terme source S. La loi de conservation de ϕ s'énonce ainsi :

$$\frac{\partial \phi}{\partial t} + \mathrm{div}(\phi \mathbf{u}) = S$$

On énonce les premières lois de conservation en deux dimensions (on note alors $\mathbf{u} = (u_x, u_y)$ en coordonées cartésiennes ou $\mathbf{u} = (u_r, u_\theta)$ en coordonnées cylindriques) :

Masse volumique ρ

$$\frac{\partial \rho}{\partial t} + \mathrm{div}(\rho \mathbf{u}) = 0$$

- Coordonnées cartésiennes : $\dfrac{\partial \rho}{\partial t} + \dfrac{\partial \rho u_x}{\partial x} + \dfrac{\partial \rho u_y}{\partial y} = 0$
- Coordonnées cylindriques : $\dfrac{\partial \rho}{\partial t} + \dfrac{1}{r}\dfrac{\partial r \rho u_r}{\partial r} + \dfrac{1}{r}\dfrac{\partial \rho u_\theta}{\partial \theta} = 0$

Quantité de mouvement $\rho\mathbf{u}$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \mathrm{div}(\rho \mathbf{u} \otimes \mathbf{u}) = -\mathbf{grad}(p)$$

- Coordonnées cartésiennes : $\begin{cases} \dfrac{\partial \rho u_x}{\partial t} + \dfrac{\partial \rho u_x^2}{\partial x} + \dfrac{\partial \rho u_x u_y}{\partial y} = -\dfrac{\partial p}{\partial x} \\ \dfrac{\partial \rho u_y}{\partial t} + \dfrac{\partial \rho u_x u_y}{\partial x} + \dfrac{\partial \rho u_y^2}{\partial y} = -\dfrac{\partial p}{\partial y} \end{cases}$
- Coordonnées cylindriques : $\begin{cases} \dfrac{\partial \rho u_r}{\partial t} + \dfrac{1}{r}\dfrac{\partial r \rho u_r^2}{\partial r} + \dfrac{1}{r}\dfrac{\partial \rho u_r u_\theta}{\partial \theta} = -\dfrac{\partial p}{\partial r} \\ \dfrac{\partial \rho u_\theta}{\partial t} + \dfrac{1}{r}\dfrac{\partial r \rho u_r u_\theta}{\partial r} + \dfrac{1}{r}\dfrac{\partial \rho u_\theta^2}{\partial \theta} = -\dfrac{\partial p}{\partial \theta} \end{cases}$

Le système d'équations fluides est infini. L'équation suivante, par exemple, donne la conservation de l'énergie. On remarque que chaque équation fait intervenir des variables déterminées dans l'équation suivante. Ainsi, il faut fermer le système d'équations avec une équation de fermeture donnée par la physique du problème (écoulement isentropique, adiabatique, etc.)

B.2 Champs de vitesse

$$u'_x = \frac{\partial \psi}{\partial y}, \quad u'_y = -\frac{\partial \psi}{\partial x}.$$

Pour u_x

$$\frac{\partial \psi}{\partial y} = \frac{b}{2\pi} R \exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right) \frac{\partial}{\partial y}\left(\frac{1}{2} - \frac{r^2}{2R^2}\right).$$

Or,

$$\frac{\partial}{\partial y}\left(\frac{1}{2} - \frac{r^2}{2R^2}\right) = -\frac{y - y_c}{R^2}$$

d'où

$$u'_x = -\frac{b}{2\pi}\frac{y - y_c}{R} \exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right).$$

Finalement,

$$\boxed{u_x = u_{x,\infty} - \frac{b}{2\pi}\frac{y-y_c}{R}\exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right)}$$

Pour u_y

$$-\frac{\partial\psi}{\partial x} = -\frac{b}{2\pi}R\exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right)\frac{\partial}{\partial x}\left(\frac{1}{2} - \frac{r^2}{2R^2}\right)$$

avec

$$\frac{\partial}{\partial x}\left(\frac{1}{2} - \frac{r^2}{2R^2}\right) = -\frac{x-x_c}{R^2}$$

ce qui conduit à

$$u'_y = \frac{b}{2\pi}\frac{x-x_c}{R}\exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right).$$

Finalement,

$$\boxed{u_y = u_{y,\infty} + \frac{b}{2\pi}\frac{x-x_c}{R}\exp\left(\frac{1}{2} - \frac{r^2}{2R^2}\right)}$$

B.3 Champs de température et de densité

Injection dans les équations d'Euler Dans toute la suite, on se place dans le référentiel du centre du tourbillon, de sorte à considérer uniquement les perturbations aux champs de vitesses initiaux. On utilise les coordonnées cylindriques, les mieux adaptées au problème. On considère un phénomène stationnaire.

Conservation de la masse volumique

$$\frac{1}{r}\frac{\partial r\rho u_r}{\partial r} + \frac{1}{r}\frac{\partial \rho u_\theta}{\partial \theta} = 0 \qquad \Longrightarrow \rho\frac{\partial u_\theta}{\partial\theta} + u_\theta\frac{\partial\rho}{\partial\theta} = 0 \qquad \Longrightarrow \frac{\partial\rho}{\partial\theta} = 0$$

Conservation de l'impulsion selon r

$$\frac{1}{r}\frac{\partial r\rho u_r^2}{\partial r} + \frac{1}{r}\frac{\partial \rho u_r u_\theta}{\partial\theta} = -\frac{\partial p}{\partial r} \qquad \Longrightarrow u_r\frac{\partial u_r}{\partial r} + \frac{u_\theta}{r}\frac{\partial u_r}{\partial\theta} - \frac{u_\theta^2}{r} = -\frac{1}{\rho}\frac{\partial p}{\partial r} \qquad \Longrightarrow \frac{1}{\rho}\frac{\partial p}{\partial r} = \frac{u_\theta^2}{r}$$

Conservation de l'impulsion selon θ

$$\frac{1}{r}\frac{\partial r\rho u_r u_\theta}{\partial r} + \frac{1}{r}\frac{\partial \rho u_\theta^2}{\partial\theta} = -\frac{\partial p}{\partial\theta} \qquad \Longrightarrow u_r\frac{\partial u_\theta}{\partial r} + \frac{u_\theta}{r}\frac{\partial u_\theta}{\partial\theta} + \frac{u_r u_\theta}{r} = -\frac{1}{\rho r}\frac{\partial p}{\partial\theta} \qquad \Longrightarrow \frac{\partial p}{\partial\theta} = 0$$

Température et densité On a utilisé les équations d'Euler pour les deux premiers moments de la fonction de distribution des vitesses, ce qui se traduit par la conservation de la masse volumique et de l'impulsion.

On a besoin d'une équation de fermeture pour résoudre le système. On utilise le caractère isentropique du tourbillon. L'équation d'état du gaz parfait s'écrit $p = \rho R_m T$ (où R_m est la constante des gaz parfaits) et la loi de Laplace donne $\frac{p}{\rho^\gamma} = \frac{p_\infty}{\rho_\infty^\gamma}$.

En prenant le logarithme dans la loi de Laplace et en dérivant, on obtient $\frac{\partial\rho}{\partial r} = \frac{1}{\gamma}\frac{\rho}{p}\frac{\partial p}{\partial r}$. On dérive ensuite l'équation d'état :

$$\frac{\partial p}{\partial r} = R_m T\frac{\partial\rho}{\partial r} + R_m\rho\frac{\partial T}{\partial r} \qquad \Longrightarrow \frac{1}{\rho}\frac{\partial p}{\partial r} = \frac{R_m}{\gamma}\frac{T}{p}\frac{\partial p}{\partial r} + R_m\frac{\partial T}{\partial r} \qquad \Longrightarrow \frac{\partial p}{\partial r}\left(\frac{1}{\rho} - \frac{1}{\gamma\rho}\right) = R_m\frac{\partial T}{\partial r}$$

En introduisant $c_P = \frac{R_m\gamma}{\gamma-1}$, on obtient $\frac{1}{\rho}\frac{\partial p}{\partial r} = c_P\frac{\partial T}{\partial r}$.

De là, on a $\dfrac{\partial T}{\partial r} = \dfrac{1}{c_P}\dfrac{u_\theta^2}{r} = \dfrac{b^2}{4\pi^2 c_P}\dfrac{r}{R^2}\exp\left(1-\dfrac{r^2}{R^2}\right)$. La solution de cette équation différentielle est

$$T(r) = T_\infty - \frac{b^2}{8\pi^2 c_P}\exp\left(1-\frac{r^2}{R^2}\right)$$

avec $T_\infty = \dfrac{b^2 e}{8\pi^2 c_P}$.

Puis, d'après la loi de Laplace, $p = \dfrac{\rho^\gamma}{\rho_\infty^\gamma}p_\infty = \rho R_m T$, d'où $\rho^{\gamma-1} = \rho_\infty^{\gamma-1}\dfrac{\rho_\infty}{p_\infty}R_m T$. On a donc

$$\rho(r) = \rho_\infty\left(\underbrace{\frac{\rho_\infty}{p_\infty}R_m T_\infty}_{1} - \frac{\rho_\infty}{p_\infty}\frac{b^2}{8\pi^2}\frac{\gamma-1}{\gamma}\exp\left(1-\frac{r^2}{R^2}\right)\right)^{\frac{1}{\gamma-1}}$$

C Algorithme GradNorm

Algorithm 1 Entraînement avec GradNorm

Initialiser $w_i(0) = 1$ pour tout i
Initialiser les poids du réseau W
Choisir un hyperparamètre $\alpha > 0$
for $t = 0$ à `max_train_steps` **do**
    Entrée d'un batch x_i
    Calculer les pertes $L_i(t)$ pour tout i, puis la perte totale : $L(t) = \sum_i w_i(t) L_i(t)$
    Calculer $G^{(i)}(t)$ et $r_i(t)$ pour tout i
    Calculer $\overline{G}(t) = \frac{1}{N} \sum_i G^{(i)}(t)$
    Calculer $\mathcal{L}_{\text{grad}}(t) = \sum_i \left\| G^{(i)}(t) - \overline{G}(t) \cdot [r_i(t)]^{\alpha} \right\|_1$
    Calculer les gradients $\nabla_{w_i} \mathcal{L}_{\text{grad}}$ (en gardant les cibles fixes)
    Calculer les gradients standards $\nabla_W L(t)$
    Mettre à jour $w_i(t) \leftarrow w_i(t+1)$ en utilisant $\nabla_{w_i} \mathcal{L}_{\text{grad}}$
    Mettre à jour $W(t) \leftarrow W(t+1)$ en utilisant $\nabla_W L(t)$
    Renormaliser les $w_i(t+1)$ pour que $\sum_i w_i(t+1) = T$
end for

D PINNs pour les géométries complexes

D.1 Prescription des conditions de bord

Les modèles numériques classiques, comme les méthodes d'éléments finis, perdent souvent en efficacité lorsque le domaine de résolution a une géométrie trop complexe. Les PINNs présentent une alternative intéressante à ces modèles.

La fonction de coût d'un PINN comporte un terme de conditions de bord, qui pénalise les solutions ne respectant pas les prescriptions physiques sur la frontière du domaine. Nous avons étudié comment, en intégrant ce terme de bord directement dans l'ansatz fournie au réseau de neurones, il est possible d'imposer efficacement ces conditions de bord.

Nous nous intéressons au problème (stationnaire pour simplifier) suivant :

$$\begin{cases} \mathcal{L}u(\mathbf{x}) = f(\mathbf{x}) & \forall \mathbf{x} \in \Omega \\ \mathcal{B}u(\mathbf{x}) = g(\mathbf{x}) & \forall \mathbf{x} \in \Gamma \subset \partial\Omega \end{cases}$$

où $\mathcal{L}$ et $\mathcal{B}$ sont des opérateurs différentiels, f est une fonction source et g est la donnée du bord. On construit une fonction de coût pertinente, par exemple avec la norme L^2 :

$$\mathcal{C}(\hat{u}) = \frac{1}{2}\|\mathcal{L}\hat{u} - f\|_{L^2}^2 = \frac{1}{2}\int_\Omega \|\mathcal{L}\hat{u} - f\|_2^2$$

où $\hat{u}$ est l'ansatz calculée par le réseau de neurones. Après avoir discrétisé Ω et Γ en deux ensembles de points Ω_d et Γ_d, trouver les paramètres optimaux du PINN revient à résoudre le problème d'optimisation suivant :

$$w^*, b^* = \arg\min_{w,b} \frac{1}{2}\frac{1}{|\Omega_d|}\sum_{\mathbf{x}\in\Omega_d} \|\mathcal{L}\hat{u}(\mathbf{x}) - f(\mathbf{x})\|_2^2$$

sous les contraintes

$$\mathcal{B}\hat{u}(\mathbf{x}) = g(\mathbf{x}) \quad \forall \mathbf{x} \in \Gamma_d$$

D.1.1 Intégration des conditions de bord dans la fonction de coût

Un premier moyen de résoudre le problème d'optimisation précédent est d'intégrer les conditions de bord dans un nouveau terme de la fonction de coût, qui se réécrit :

$$\mathcal{C}(\hat{u}) = \frac{1}{2}\frac{1}{|\Omega_d|}\sum_{\mathbf{x}\in\Omega_d} \|\mathcal{L}\hat{u}(\mathbf{x}) - f(\mathbf{x})\|_2^2 + \frac{1}{2}\frac{1}{|\Gamma_d|}\sum_{\mathbf{x}\in\Gamma_d} \|\mathcal{B}\hat{u}(\mathbf{x}) - g(\mathbf{x})\|_2^2$$

Cette approche donne en fait un PINN, comme nous l'avons vu précédemment.

Cependant, ajouter ce terme dans la fonction de coût peut engendrer des instabilités et des problèmes de convergence, en particulier dans des géométries complexes ou en grandes dimensions. C'est pourquoi nous présentons des approches alternatives, promettant un meilleur comportement.

D.1.2 Intégration des conditions de bord dans l'ansatz

Au lieu d'intégrer les conditions de bord dans la formulation de la fonction de coût, on peut construire une ansatz $\hat{u}$ qui respectera nécessairement ces conditions, puis résoudre le problème d'optimisation sans contraintes avec cette ansatz. Cette approche a été exposée dans [1]. À chaque nouvelle utilisation du réseau de neurones, on utilise donc l'ansatz suivante (nous nous plaçons ici dans le cadre des conditions de Dirichlet) :

$$\hat{u}(\mathbf{x}) = G(\mathbf{x}) + D(\mathbf{x})\hat{u}^L(\mathbf{x})$$

Ici, G est une extension régulière de la donnée du bord g sur tout $\partial\Omega$, et D est une fonction régulière donnant la distance de $\mathbf{x}$ à Γ.

Cette ansatz présente l'avantage évident (c'est ce pour quoi nous l'avons construite) de respecter les conditions de bord : $\forall \mathbf{x} \in \Gamma, D(\mathbf{x}) = 0$ et donc $\hat{u}(\mathbf{x}) = G(\mathbf{x}) = g(\mathbf{x})$.

Nous expliquons ensuite comment calculer G et D à l'aide de réseaux de neurones :

Calcul de G On impose que G soit suffisamment proche de g sur Γ :

$$\|G(\mathbf{x}) - g(\mathbf{x})\|_2 < \epsilon \quad \forall \mathbf{x} \in \Gamma$$

On peut donc calculer G en utilisant un réseau de neurones avec la fonction de coût suivante :

$$\mathcal{C}(G) = \frac{1}{2} \frac{1}{|\Gamma_d|} \sum_{\mathbf{x} \in \Gamma_d} \|G(\mathbf{x}) - g(\mathbf{x})\|_2^2$$

Calcul de D On impose que D soit petite sur Γ et proche de la fonction de distance à Γ analytique sur tout Ω :

$$\|D(\mathbf{x})\|_2 < \epsilon \quad \forall \mathbf{x} \in \Gamma, \qquad D(\mathbf{x}) \approx d(x) := \min_{\mathbf{y} \in \Gamma} \|\mathbf{x} - \mathbf{y}\|_2 \quad \forall \mathbf{x} \in \Omega$$

On peut donc calculer D en utilisant un réseau de neurones avec la fonction de coût suivante :

$$\mathcal{C}(D) = \frac{1}{2} \frac{1}{|\omega_d| + |\Gamma_d|} \sum_{\mathbf{x} \in \omega_d \cup \Gamma_d} \|D(\mathbf{x}) - d(\mathbf{x})\|_2^2$$

avec $\omega_d \subset \Omega_d$.

Si $|\omega_d| + |\Gamma_d| \ll |\Omega_d|$, le coût computationnel de G et D est négligeable devant le coût de résolution du problème.

D.1.3 Prescription exacte des conditions de bord

Pour obtenir plus de précision dans l'approche précédente, il est possible de calculer analytiquement la fonction de distance D. C'est l'approche qui a été développée dans [6]. Plus précisément, on cherche à approximer analytiquement la fonction de distance théorique $d : \mathbf{x} \mapsto \min_{\mathbf{y} \in \Gamma} \|\mathbf{x} - \mathbf{y}\|_2$. Nous utilisons une fonction de distance approximée ϕ qui possède les propriétés suivantes :

- $\phi(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \Gamma$
- $\phi(\mathbf{x}) > 0 \quad \forall \mathbf{x} \in \Omega$
- $\boldsymbol{\nabla} \phi(\mathbf{x}) \neq 0 \quad \forall \mathbf{x} \in \Gamma$
- $\dfrac{\partial \phi}{\partial \mathbf{v}} = 1, \quad \dfrac{\partial^k \phi}{\partial \mathbf{v}^k} = 0 \quad \forall k \in [\![2, m]\!]$ avec $\mathbf{v}$ le vecteur unitaire normal à Γ et m l'ordre de normalisation de ϕ.

On pose ensuite une ansatz de la forme $\hat{u}(\mathbf{x}) = G(\mathbf{x}) + \phi(\mathbf{x}) \hat{u}^L(\mathbf{x})$.

Le calcul de la fonction de distance approximée repose sur l'utilisation des R-fonctions, qui permettent de représenter la géométrie d'un domaine. Une R-fonction élémentaire f_A associée à un ensemble A est positive à l'intérieur de A et négative à l'extérieur. Il est de plus possible de combiner des R-fonctions entre elles pour représenter des géométries de plus en plus variées. Nous considérons ici deux ensembles A et B et les R-fonctions élémentaires associées f_A et f_B :

Négation (complémentaire) La R-fonction représentant $\bar{A}$ est $\neg f_A = -f_A$.

Disjonction (union) La R-fonction représentant $A \cup B$ est $f_A \vee f_B = \max(f_A, f_B)$.

Conjonction (intersection) La R-fonction représentant $A \cap B$ est $f_A \wedge f_B = \min(f_A, f_B)$.

L'équivalence entre R-fonctions permet de déterminer la fonction de distance approximée à une frontière composée de plusieurs éléments, en connaissance des fonctions de distance approximées de chaque élément :

$$\phi(\phi_1, \ldots, \phi_n) = \frac{1}{\left(\frac{1}{\phi_1^m} + \ldots + \frac{1}{\phi_n^m}\right)^{\frac{1}{m}}}$$

Nous présentons la construction analytique de fonctions de distance approximées dans trois cas simples en deux dimensions : le segment, l'arc de cercle et le triangle.

On notera dans la suite $\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$ pour tout point $\mathbf{x}$ du plan.

Segment On considère un segment $[\mathbf{x}_1, \mathbf{x}_2]$. On note $L = \|\mathbf{x}_2 - \mathbf{x}_1\|_2$ la longueur du segment et $\mathbf{x}_c = \dfrac{\mathbf{x}_1 + \mathbf{x}_2}{2}$ le milieu du segment.

La distance signée à la droite passant par $\mathbf{x}_1$ et $\mathbf{x}_2$ depuis un point $\mathbf{x}$ est :

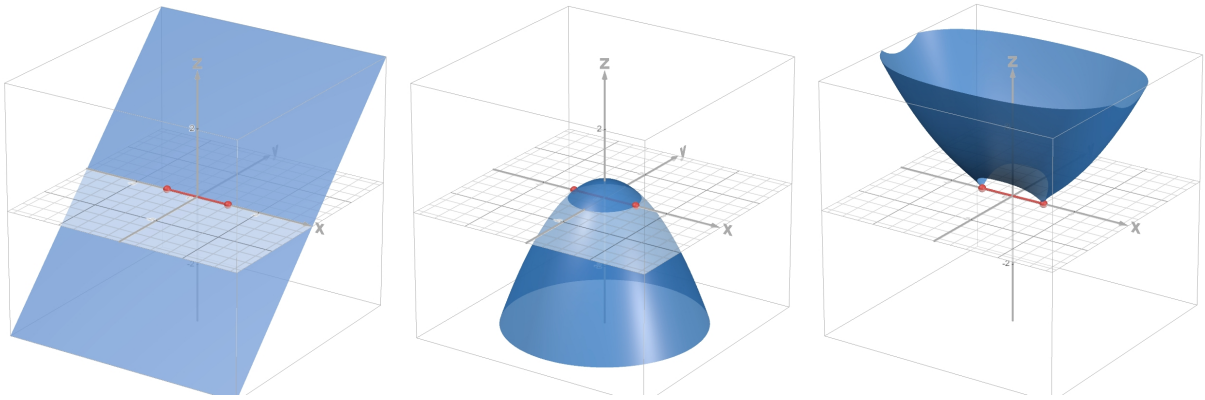
$$d(\mathbf{x}) = \frac{(x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1)}{L}$$

On définit ensuite la fonction de *trimming* :

$$t(\mathbf{x}) = \frac{1}{L}\left(\left(\frac{L}{2}\right)^2 - \|\mathbf{x} - \mathbf{x}_c\|_2^2\right)$$

La fonction de distance approximée est alors donnée par

$$\boxed{\phi(\mathbf{x}) = \sqrt{f(\mathbf{x})^2 + \left(\frac{\sqrt{t(\mathbf{x})^2 + f(\mathbf{x})^4} - t(\mathbf{x})}{2}\right)^2}}$$



(a) d (b) t (c) ϕ

Figure 10: Fonction de distance approximée pour le segment $\left[\begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}\right]$

Arc de cercle On considère l'arc de cercle de centre $\mathbf{x}_c$ et de rayon R, défini paramétriquement par

$$\mathbf{x}(\theta) = \mathbf{x}_c + R\begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \quad \forall \theta \in [\theta_1, \theta_2]$$

On note $\mathbf{x}_1 = \mathbf{x}_c + R\begin{pmatrix}\cos(\theta_1)\\ \sin(\theta_1)\end{pmatrix}$ et $\mathbf{x}_2 = \mathbf{x}_c + R\begin{pmatrix}\cos(\theta_2)\\ \sin(\theta_2)\end{pmatrix}$.

La distance au cercle superposé à l'arc de cercle depuis un point $\mathbf{x}$ est

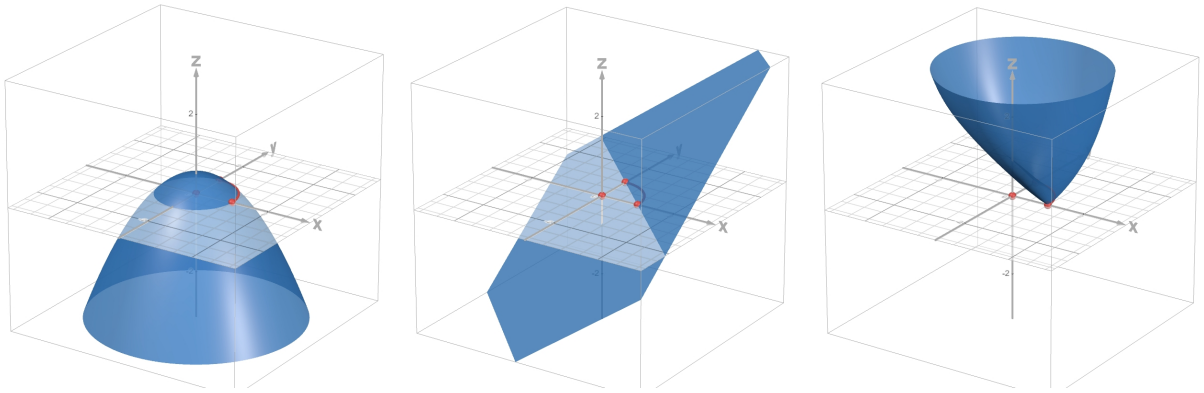
$$d(\mathbf{x}) = \frac{R^2 - \|\mathbf{x} - \mathbf{x}_c\|_2^2}{2R}$$

On définit, comme précédemment, la fonction de *trimming* :

$$d(\mathbf{x}) = \frac{(x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1)}{\|\mathbf{x}_2 - \mathbf{x}_1\|_2}$$

La fonction de distance approximée est alors donnée par

$$\boxed{\phi(\mathbf{x}) = \sqrt{f(\mathbf{x})^2 + \left(\frac{\sqrt{t(\mathbf{x})^2 + f(\mathbf{x})^4} - t(\mathbf{x})}{2}\right)^2}}$$



(a) d (b) t (c) ϕ

Figure 11: Fonction de distance approximée pour l'arc de cercle $\begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}\cos(\theta)\\ \sin(\theta)\end{pmatrix}, \forall\theta \in \left[0, \frac{\pi}{2}\right]$

Triangle En utilisant la relation d'équivalence des R-fonctions

$$\phi(\phi_1, \phi_2, \phi_3) = \frac{1}{\left(\frac{1}{\phi_1^m} + \frac{1}{\phi_2^m} + \frac{1}{\phi_3^m}\right)^{\frac{1}{m}}}$$

on peut facilement construire la fonction de distance approximée d'un triangle à partir des fonctions de distance approximées de ses trois côtés.

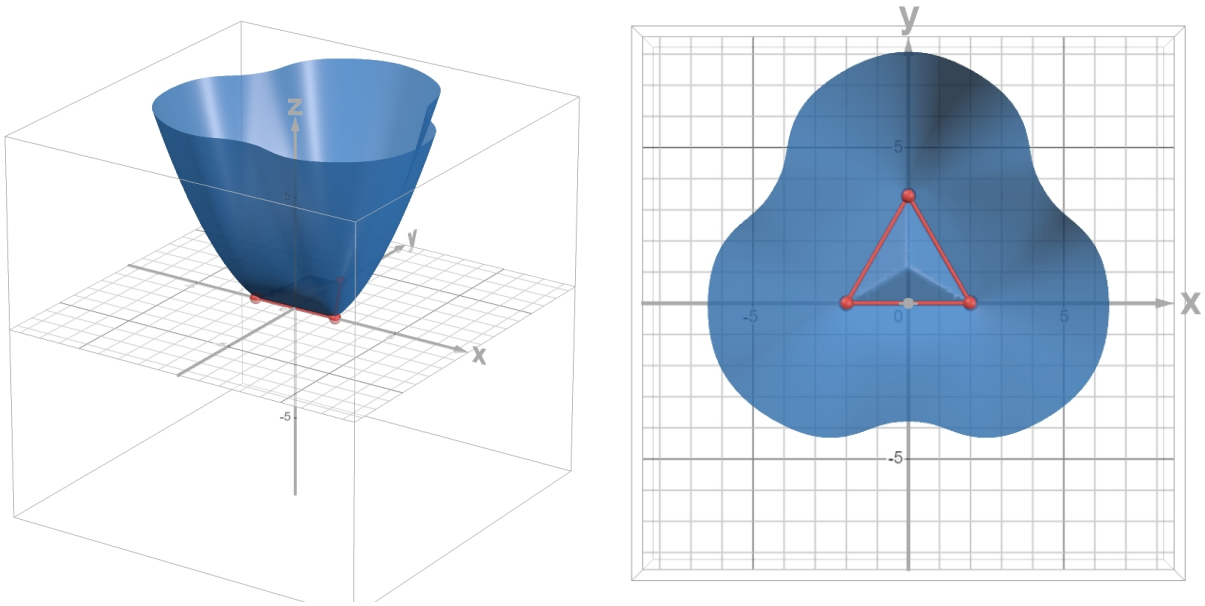


Figure 12: Fonction de distance approximée pour le triangle équilatéral

D.2 Prescription de la topologie

Les géométries complexes peuvent l'être en raison de conditions de bord spécifiques comme nous l'avons vu, mais également à cause d'une topologie particulière. Nous présentons une approche développée dans [4] qui traite de ces topologies complexes en introduisant les GPINNs.

Les PINNs classiques opèrent dans des espaces euclidiens ce qui ne correspond pas toujours aux contraintes physiques du problème. Le GPINN prend en compte la topologie du problème en introduisant une nouvelle variable z à la fonction approximée, qui devient $\hat{u}(t, \mathbf{x}, z)$. Cette dimension supplémentaire caractérise la géométrie particulière du domaine de résolution considéré.

Comment déterminer z et en quoi cette variable est-elle liée à la topologie du problème ? On utilise un graphe $G = (V, E)$ pour encoder la géométrie du domaine. Ici, V contient les noeuds du graphe et E les arêtes. On note A la matrice d'adjacence du graphe. On définit alors la matrice diagonale D :

$$D_{ii} = \sum_j A_{ij}$$

On appelle vecteur de Fielder le second vecteur propre de la matrice $D - A$. La variable z est le vecteur de Fielder.

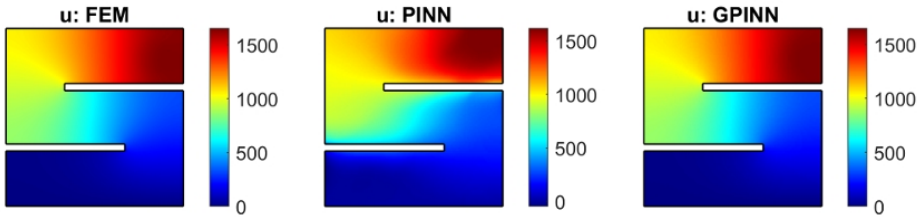
Nous donnons dans la suite deux exemples permettant d'évaluer l'efficacité des GPINNs dans deux cas physiques en deux dimensions : la propagation de chaleur et la fissure. On comparera dans chaque cas la solution donnée par la méthode des éléments finis, celle obtenue avec un PINN classique et celle du GPINN.

Propagation de la chaleur L'équation de la chaleur stationnaire est

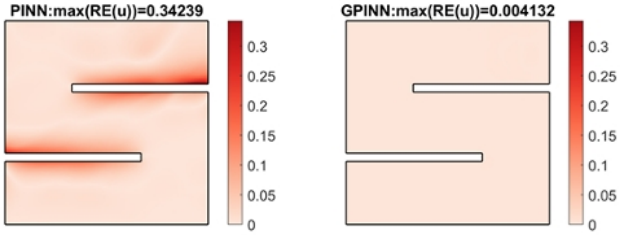
$$\Delta u(\mathbf{x}) = f(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega$$

avec des conditions de bord

$$\begin{cases} u(\mathbf{x}) = u_B(\mathbf{x}) & \forall \mathbf{x} \in \partial\Omega_D \\ \boldsymbol{\nabla} u(\mathbf{x}) \cdot \mathbf{n} = v(\mathbf{x}) & \forall \mathbf{x} \in \partial\Omega_N \end{cases}$$



(a) Solution obtenue



(b) Erreur par-rapport à la solution analytique

Figure 13: Résultats pour le problème de propagation de la chaleur
(Crédit : [4])

Les résultats obtenus avec le PINN classique sont moins satisfaisants que ceux obtenus avec le GPINN en raison de la géométrie complexe du domaine.

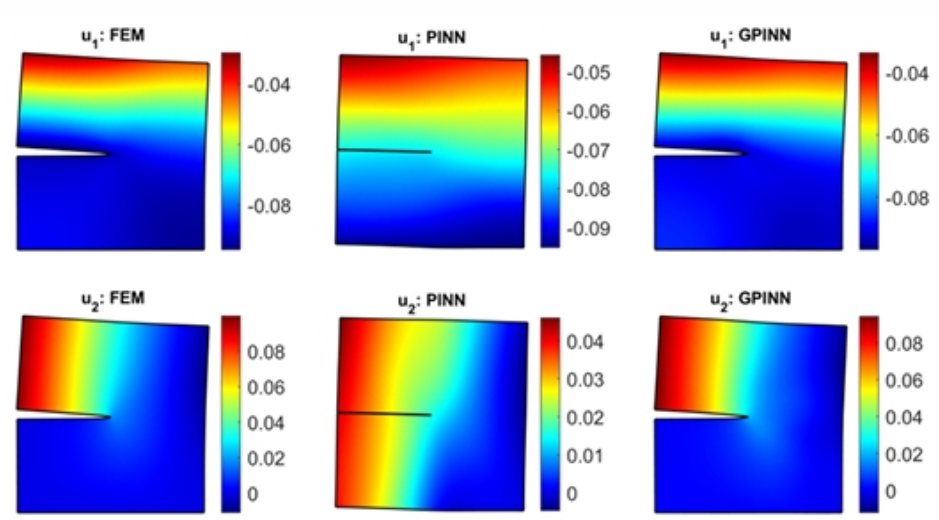
Fissure L'équation d'équilibre en élasticité linéaire s'écrit :

$$\boldsymbol{\nabla} \cdot \sigma(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \Omega$$

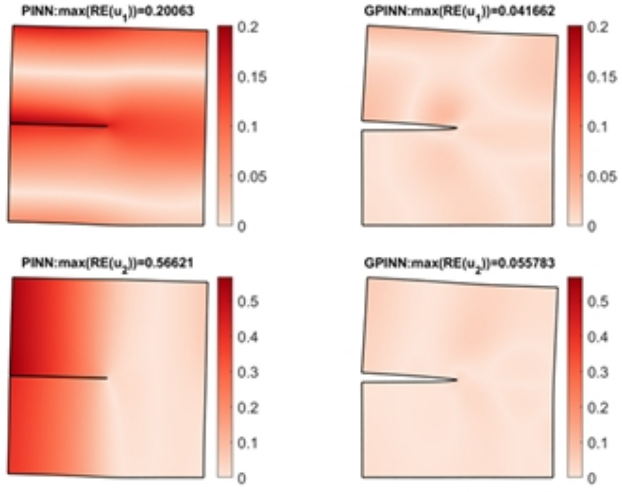
avec les conditions de bord

$$\begin{cases} \sigma(\mathbf{x}) \cdot \mathbf{n} = \mathbf{t}(\mathbf{x}) & \forall \mathbf{x} \in \partial\Omega_N \\ \mathbf{u}(\mathbf{x}) = \mathbf{u}_B(\mathbf{x}) & \forall \mathbf{x} \in \partial\Omega_D \end{cases}$$

Rappelons que l'on a la relation $\sigma(\mathbf{x}) = C : \epsilon(\mathbf{x})$ avec $\epsilon(\mathbf{x}) = \boldsymbol{\nabla}\mathbf{u}(\mathbf{x})$.



(a) Solution obtenue



(b) Erreur par-rapport à la solution analytique

Figure 14: Résultats pour le problème de fissure
(Crédit : [4])

Là encore, on obtient des résultats bien meilleurs avec le GPINN, qui prend en compte la topologie du problème, qu'avec le PINN classique qui présente des résultats très mauvais sur une partie du domaine.

Ainsi, ces techniques permettent de modéliser correctement des problèmes géométriquement complexes ou présentent des discontinuités importantes.

Références

[1] Jens Berg et Kaj Nyström. "A unified deep artificial neural network approach to partial differential equations in complex geometries". In : *Neurocomputing* 317 (23 août 2018), p. 28-41. doi : 10.1016/j.neucom.2018.06.056. url : https://doi.org/10.1016/j.neucom.2018.06.056.

[2] Zhao Chen et al. "GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks". In : *International Conference on Machine Learning* (3 juill. 2018), p. 794-803. url : http://proceedings.mlr.press/v80/chen18a/chen18a.pdf.

[3] A. Ali Heydari, Craig A. Thompson et Asif Mehmood. "SoftAdapt: Techniques for Adaptive Loss Weighting of Neural Networks with Multi-Part Loss Functions". In : *arXiv (Cornell University)* (1er jan. 2019). doi : 10.48550/arxiv.1912.12355. url : https://arxiv.org/abs/1912.12355.

[4] Yuyang Miao, Haolin Li et Danilo Mandic. "GPINN: Physics-Informed Neural Network with Graph Embedding". In : *2022 International Joint Conference on Neural Networks (IJCNN)* (30 juin 2024), p. 1-8. doi : 10.1109/ijcnn60899.2024.10651053. url : https://doi.org/10.1109/ijcnn60899.2024.10651053.

[5] M. Raissi, P. Perdikaris et G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In : *Journal of Computational Physics* 378 (3 nov. 2018), p. 686-707. doi : 10.1016/j.jcp.2018.10.045. url : https://doi.org/10.1016/j.jcp.2018.10.045.

[6] N. Sukumar et Ankit Srivastava. "Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks". In : *Computer Methods in Applied Mechanics and Engineering* 389 (2 déc. 2021), p. 114333. doi : 10.1016/j.cma.2021.114333. url : https://doi.org/10.1016/j.cma.2021.114333.

[7] Gauthier Wissocq, Jean-François Boussuge et Pierre Sagaut. "Consistent vortex initialization for the athermal lattice Boltzmann method". In : *Physical review. E* 101.4 (23 avr. 2020). doi : 10.1103/physreve.101.043306. url : https://doi.org/10.1103/physreve.101.043306.