# Codev Recherche : Dissipative Hamiltonian Neural Networks

**Mohammed El Hassan Ayoubi**
**Sous supérvision de Lucas Drumetz**

## Abstract

Comprendre les symétries naturelles est essentiel pour donner un sens à notre monde complexe et en constante évolution. La mécanique hamiltonienne, une théorie bien établie pour modéliser l'évolution temporelle des systèmes avec des quantités conservées (telles que l'énergie totale), a récemment été étendue par l'utilisation de modèles d'apprentissage automatique, notamment des réseaux neuronaux hamiltoniens (HNNs). Cependant, les HNNs rencontrent des difficultés avec des ensembles de données bruitées et éparses, et peinent particulièrement lorsque l'énergie n'est pas conservée. Dans cet article, nous nous demandons s'il est possible d'identifier et de décomposer simultanément des dynamiques conservatrices et dissipatives. Nous proposons les Réseaux Neuronaux Hamiltoniens Dissipatifs (D-HNNs), qui paramètrent à la fois une fonction hamiltonienne et une fonction de dissipation de Rayleigh. De plus, nous introduisons les Processus Gaussiens Spectraux Symplectiques (SSGPs), intégrant la structure géométrique symplectique des systèmes hamiltoniens comme distribution a priori pour estimer ces systèmes avec dissipation additive. Cette approche permet de construire un algorithme d'inférence variationnelle efficace pour entraîner les modèles tout en simulant les dynamiques via des solveurs d'équations différentielles ordinaires. Les expériences menées sur plusieurs systèmes physiques démontrent que notre approche offre d'excellentes performances pour prédire les dynamiques suivant les lois de conservation ou de dissipation de l'énergie, à partir de données bruitées et éparses.

## Introduction

Malgré les grands progrès réalisés dans la simulation des problèmes multiphysiques en utilisant la discrétisation numérique des équations aux dérivées partielles (EDP), on ne peut toujours pas incorporer de manière transparente des données bruyantes dans les algorithmes existants, la génération de maillages reste complexe et les problèmes de haute dimension régis par des EDP paramétrées ne peuvent pas être abordés. De plus, la résolution des problèmes inverses avec des phénomènes physiques cachés est souvent prohibitive en termes de coût et nécessite des formulations différentes et des codes informatiques élaborés.

L'apprentissage automatique a émergé comme une alternative prometteuse (1), mais l'entraînement des réseaux de neurones profonds nécessite de grandes quantités de données, qui ne sont pas toujours disponibles pour les problèmes scientifiques. En revanche, ces réseaux peuvent être entraînés à partir d'informations supplémentaires obtenues en imposant les lois physiques (par exemple, à des points aléatoires dans le domaine espace-temps continu). Cet apprentissage informé par la physique intègre des données (bruyantes) et des modèles mathématiques, et les implémente à travers des réseaux de neurones ou d'autres réseaux de régression.

Dans ce contexte, notre projet se concentre spécifiquement sur l'intégration des principes de conservation et de dissipation de l'énergie dans les modèles de réseaux de neurones et autres types de modèles d'apprentissage automatique. Nous passons en revue certaines des tendances dominantes dans l'intégration de la physique dans l'apprentissage automatique, présentons certaines des capacités et limitations actuelles, et discutons des diverses applications de l'apprentissage informé par la physique. Nous allons nous restreindre, dans tout ce travail, au cas du pendule simple.

## 1. Méthodes

### 1.1. Mécanique Hamiltonienne

Dans cette section, Mécanique Hamiltonienne (5) ,nous allons faire un petit tour de la mécanique hamiltonienne.

On considère un système avec $N$ degrés de liberté. Dans le formalisme hamiltonien, l'évolution du système en temps continu est décrite dans l'espace des phases, c'est-à-dire l'espace produit scalaire des coordonnées généralisées $x^q = (x_1^q, \dots, x_N^q)$ et des moments généralisés $x^p = (x_1^p, \dots, x_N^p)$.

Soit $x = (x^q, x^p) \in \mathbb{R}^D$ l'état du système, avec $D = 2N$.

L'évolution du système est déterminée par le hamiltonien $H(x) : \mathbb{R}^D \to \mathbb{R}$ (3) , qui représente l'énergie totale du système.

La modélisation du système avec la composante dissipative est donnée par :

$$\frac{dx}{dt} = (S - R)\nabla H(x) =: f(x) \quad (1)$$

avec

$$S = \begin{pmatrix} O & I \\ -I & O \end{pmatrix},$$

où $\nabla H(x) : \mathbb{R}^D \to \mathbb{R}^D$ désigne le gradient du hamiltonien par rapport à $x$, $S \in \mathbb{R}^{D\times D}$ est une matrice antisymétrique, $R \in \mathbb{R}^{D\times D}$ est la matrice dissipative semi-définie positive, $I$ est la matrice identité, et $O$ est la matrice nulle.

Dans (1), on définit la dérivation temporelle du système par la fonction $f(x) : \mathbb{R}^D \to \mathbb{R}^D$, qui est un type spécial de champ vectoriel ayant une structure géométrique symplectique (9)(appelé champ vectoriel hamiltonien ou gradient symplectique). La dynamique sur ce champ vectoriel conserve l'énergie totale lorsque $\mathbf{R} = \mathbf{O}$. Un exemple de matrice dissipative est $\mathbf{R} = \text{diag}(0, \ldots, 0, r_1, \ldots, r_N)$, représentant un système dissipatif avec un coefficient de frottement $r_i \geq 0$ pour $i = 1, \ldots, N$.

Bien que nous supposions ce type de matrice de dissipation dans ce qui suit, ce travail est extensible à une matrice de dissipation générale, telle que le terme d'amortissement dépendant de l'état.

Ayant accès à un champ de vecteurs $\mathbf{f}(\mathbf{x})$ et une condition $x_1$ à $t_1$, on peut prédire $x_t$ à $t$ en intégrant $\mathbf{f}(\mathbf{x})$ entre $t_1$ et $t$ en utilisant la formule :

$$x_t = x_1 + \int_{t_1}^{t} f(x)\, dt$$

### 1.2. Application au cas du pendule simple

Le lagrangien est : $\mathcal{L} = \frac{1}{2}ml^2\dot{\theta}^2 - mgl(1 - \cos(\theta))$. Le moment conjugué est :

$$p_\theta = \frac{\partial \mathcal{L}}{\partial \dot{\theta}} = ml^2\dot{\theta}$$

On peut donc écrire le Hamiltonien sous la forme : $\mathcal{H} = \sum_q \dot{q}p_q - \mathcal{L} = \frac{1}{2}ml^2\dot{\theta}^2 + mgl(1 - \cos(\theta)) = \frac{p_\theta^2}{2ml^2} + mgl(1 - \cos(\theta))$.

Une écriture plus simple des équations du Hamiltonien est :

$$\frac{\partial H}{\partial p} = \frac{\partial q}{\partial t}$$

$$-\frac{\partial H}{\partial q} = \frac{\partial p}{\partial t}$$

L'introduction de $H$ dans le modèle permet d'extraire la composante conservative du système.

Cependant, les systèmes réels comportent des forces dissipatives, pour cela on introduit la fonction de Rayleigh.

$$D = \frac{1}{2} \cdot \rho \cdot \dot{q}^2$$

avec $\rho$ une constante et $\dot{q}$ la vitesse.

### 1.3. Baseline model

Le modèle de référence qui, étant donné un vecteur d'entrée (q, p), produit directement le vecteur ($\partial q/\partial t$, $\partial p/\partial t$).

### 1.4. Multi Layer Perceptron

Multi Layer Perceptron (MLP) (4) est un type de réseau de neurones artificiels (ANN) (6)à propagation avant avec la structure suivante :

- **Couche d'entrée** : Reçoit les caractéristiques d'entrée.
- **Couches cachées** : Effectuent des calculs en utilisant des neurones avec des fonctions d'activation non linéaires $\sigma$.
- **Couche de sortie** : Produit la sortie finale.

#### Passage en avant

Pour un vecteur d'entrée $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, l'activation $a_j$ du $j$-ième neurone dans une couche cachée est :

$$a_j = \sigma\left(\sum_{i=1}^{n} w_{ij}x_i + b_j\right)$$

où $w_{ij}$ sont les poids et $b_j$ est le biais. Ce processus est répété pour chaque couche jusqu'à la couche de sortie.

#### Théorème d'Approximation Universelle

Le Théorème d'Approximation Universelle (2) stipule qu'un MLP avec une seule couche cachée contenant un nombre fini de neurones peut approximer toute fonction continue sur des sous-ensembles compacts de $\mathbb{R}^n$, avec une fonction d'activation appropriée.

THÉORÈME

Étant donné une fonction d'activation $\sigma$ non constante, bornée et continue, et toute fonction continue $f$ sur un sous-ensemble compact $K \subset \mathbb{R}^n$, pour tout $\epsilon > 0$, il existe des

constantes $v_i, b_i \in \mathbb{R}$ et des vecteurs $\mathbf{w}_i \in \mathbb{R}^n$ tels que :

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$$

satisfait :

$$\|f(\mathbf{x}) - F(\mathbf{x})\| < \epsilon$$

pour tout $\mathbf{x} \in K$.

Pour une fonction d'activation sigmoïde :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

La sortie d'un MLP à une seule couche est :

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$$

Les MLP sont entraînés en utilisant la rétropropagation pour minimiser une fonction de perte, ce qui leur permet d'apprendre des motifs complexes dans les données.

### 1.5. HNN

Cette approche d'HNN (5) consiste à apprendre une fonction paramétrique pour $H$. Ce faisant, nous dotons notre modèle de la capacité d'apprendre des quantités exactement conservées à partir des données de manière non supervisée.

Lors de la passe avant, il consomme un ensemble de coordonnées et produit une seule valeur scalaire de type "énergie". Ensuite, avant de calculer la perte, nous prenons le gradient in-graph de la sortie par rapport aux coordonnées d'entrée. C'est par rapport à ce gradient que nous calculons et optimisons notre fonction de coût.

$$L_{\text{HNN}} = \left\| \frac{\partial H_\theta}{\partial p} - \frac{\partial q}{\partial t} \right\| + \left\| -\frac{\partial H_\theta}{\partial q} - \frac{\partial p}{\partial t} \right\| \qquad (1)$$

Cette méthode d'entraînement permet au HNN d'apprendre des propriétés conservatives de l'énergie à partir des données.

Alors que l'objectif principal des HNN est de doter les réseaux neuronaux de meilleurs a priori physiques dans ce domaine, les HNN présentent quelques propriétés remarquables :

- Réversibilité : la plupart des Neural ODEs ne sont pas réversibles au sens mathématique : leurs mappings ne sont pas tout à fait bijectifs. Contrairement aux méthodes précédentes, les HNN sont garantis de produire des trajectoires parfaitement réversibles dans le temps.
- Variation d'énergie : jusqu'à présent, nous avons vu que l'intégration du gradient symplectique de l'Hamiltonien peut nous donner l'évolution temporelle d'un système $S_H = (\frac{\partial H}{\partial p}, -\frac{\partial H}{\partial q})$, mais nous n'avons pas essayé de suivre le gradient riemannien $R_H = (\frac{\partial H}{\partial q}, \frac{\partial H}{\partial p})$.

### 1.6. D-HNN

Les réseaux neuronaux dissipatifs hamiltoniens (D-HNN) (8) sont un modèle qui étend les HNN pour apprendre la décomposition de tout système dynamique en ses quantités dissipatives et conservées. Les D-HNN apprennent ces décompositions en apprenant une fonction de dissipation de Rayleigh $D$ en plus d'un Hamiltonien. Le réseau résultant a deux sorties scalaires, $H$ et $D$.

Le premier produit un champ de vecteurs rotationnel par rapport aux coordonnées d'entrée, tandis que le second produit un champ de vecteurs irrotationnel sur les mêmes coordonnées. En les combinant, le théorème de Hodge-Helmholtz nous dit que ces deux champs de vecteurs peuvent s'adapter à n'importe quel champ de vecteurs arbitraire. De la même manière, n'importe quel champ de vecteurs - disons, un ensemble de mesures de vélocité - peut être décomposé en la somme d'un champ de vecteurs rotationnel et d'un champ de vecteurs irrotationnel. Nous appelons cette décomposition la décomposition de Helmholtz.

En entraînant le modèle sur une base de données (produite en utilisant des données réelles ou expérimentales), il apprend de façon implicite la décomposition de Helmholtz, c'est-à-dire la partie conservatrice et dissipative de façon séparée.

La décomposition de Helmholtz est donnée par :

$$V = V_{\text{irr}} + V_{\text{rot}} = \nabla\phi + \nabla \times A$$

où $V_{\text{irr}}$ représente la composante irrotationnelle et $V_{\text{rot}}$ la composante rotationnelle.

Le modèle reçoit deux composantes $p$ et $q$ et donne une prédiction de la composante dissipative et conservatrice pour ensuite donner un champ de vecteurs.

Le modèle de D-HNN utilise deux MLP, l'un pour apprendre la composante dissipative et l'autre pour apprendre la composante conservatrice.

Étant donné que les D-HNN apprennent des décompositions exactes, ils peuvent récupérer la dynamique du système complet, qui est simplement la somme des dynamiques dissipatives et conservées. Ces décompositions permettent même de nouvelles formes de généralisation.

$$L_{\text{D-HNN}} = \left\| \left(\frac{\partial H}{\partial p} + \frac{\partial D}{\partial q}\right) - \frac{\partial q}{\partial t} \right\| + \left\| \left(-\frac{\partial H}{\partial p} + \frac{\partial D}{\partial q}\right) - \frac{\partial q}{\partial t} \right\| \tag{2}$$

Pour entraîner le modèle, les auteurs ont utilisé des données qu'ils ont produites en utilisant les équations exactes de $D$ et $H$.

Ensuite, ils ont appliqué leur modèle sur des données récupérées d'un article de recherche intitulé "Distilling Free-Form Natural Laws from Experimental Data". Ce dernier contient un fichier "real pend h 1" qui est structuré comme suit : [0, temps, position, vitesse].

### 1.7. Modèle SSGP

Dans le modèle SSGP (9), l'utilisation des processus gaussiens (GP) (10) permet de modéliser avec précision les dynamiques de systèmes physiques, en intégrant les incertitudes inhérentes aux données bruitées et éparses. Cette méthode est particulièrement efficace pour les systèmes décrits par des équations différentielles ordinaires (ODE) et utilisant la mécanique hamiltonienne.

En combinant les processus gaussiens avec les PINNs, nous exploitons les capacités des réseaux de neurones à apprendre les structures complexes et les incertitudes des GP pour intégrer des connaissances a priori sur les lois physiques. Cela permet de modéliser des systèmes où les données sont rares et bruitées, et où les dynamiques doivent respecter des lois de conservation ou de dissipation d'énergie.

Par exemple, pour prédire la trajectoire $x(t)$ à partir d'une condition initiale $x(0)$, on utilise le solveur d'ODE intégré avec les GP pour générer des trajectoires échantillons, puis on optimise les paramètres du modèle pour minimiser l'erreur entre les trajectoires observées et prédites tout en respectant les lois physiques imposées par le Hamiltonien.

Considérons un système dynamique décrit par un Hamiltonien $H(x)$, où $x = (q, p)$ représente les coordonnées généralisées et les moments généralisés. La dynamique du système est donnée par les équations de Hamilton :

$$\frac{dx}{dt} = (S - R)\nabla H(x),$$

où $S$ est une matrice antisymétrique et $R$ est une matrice de dissipation semi-définie positive.

Pour modéliser le Hamiltonien $H(x)$ avec un GP, nous posons :

$$H(x) \sim \text{GP}(0, \gamma(x, x')),$$

où $\gamma(x, x')$ est une fonction de covariance. La dynamique du système est alors modélisée par un champ de vecteurs $f(x)$ :

$$f(x) = (S - R)\nabla H(x),$$

qui est également un GP multi-sorties :

$$f(x) \sim \text{GP}(0, K(x, x')),$$

où

$$K(x, x') = (S - R)\nabla^2 \gamma(x, x')(S - R)^\top.$$

## 2. Résultats

### 2.1. Préparation des données

Afin d'entraîner le modèle, deux approches sont envisageables. La première consiste à l'entraîner sur des données exactes, c'est-à-dire des données issues de la résolution numérique des équations. Une autre approche que nous avons envisagée est de récupérer les données à partir d'une vidéo.
Cette amélioration que nous avons apportée au modèle original permet de l'entraîner à partir d'une vidéo de pendule simple.

Pour suivre le mouvement du pendule, la première idée que j'ai eue est d'utiliser les modèles déjà implémentés en Python, principalement la bibliothèque OpenCV. Cette dernière est conçue pour la détection en temps réel.

Cependant, le résultat obtenu n'était pas satisfaisant ( Figure 1).

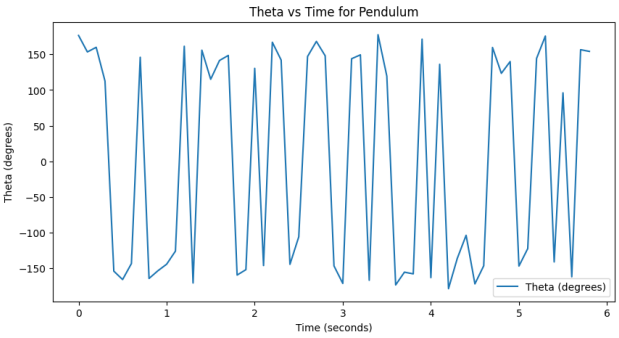


*Figure 1.* Variation de l'angle theta en fonction du temps

L'une des raisons pour lesquelles le modèle n'était pas précis est que le mouvement du pendule était très rapide. Ainsi, lorsque le modèle divise la vidéo en plusieurs images, la pendule se trouve souvent à une extrémité et n'atteint jamais une position intermédiaire, d'où la présence de pentes abruptes dans le tracé.

Afin de remédier à ce problème, j'ai opté pour un modèle de détection d'objet basé sur une architecture très performante.

YOLO signifie "You Only Look Once"; c'est une famille populaire d'algorithmes de détection d'objets en temps réel.

La version que j'ai utilisée est YOLOv7 (11). Bien que d'autres versions soient plus performantes et stables, j'ai opté pour celle-ci car elle m'a permis plus de liberté dans la récupération des "Bounding boxes".
Le modèle se compose principalement de trois grandes parties :

- Backbone
- Neck
- Head

Le Backbone extrait principalement les caractéristiques essentielles d'une image et les transmet au Head via le Neck. Le Neck collecte les cartes de caractéristiques extraites par le Backbone et crée des pyramides de caractéristiques. Enfin, le Head se compose de couches de sortie qui effectuent les détections finales.

#### 2.1.1. Résultats du modèle de détection d'objet

La réimplémentation complète du modèle s'avère difficile, c'est pourquoi j'ai utilisé une implémentation déjà présente sur GitHub et j'ai entraîné le modèle sur ma propre base de données.

Pour créer cette base de données, j'ai utilisé le site roboflow.com qui m'a permis d'annoter mes propres images et de générer une base de données compatible avec les entrées du modèle.

Pour construire cette base de données, j'ai utilisé deux vidéos différentes disponibles sur YouTube que j'ai divisées en plusieurs images et annotées en utilisant les outils de Roboflow.

Afin de rendre le modèle plus robuste et d'éviter le surapprentissage (overfitting), j'ai ajouté du bruit à chaque image et j'ai également fait pivoter certaines d'entre elles.

La base de données annotée était divisée en 88% d'entraînement, 9% de validation et 4% de test.

La répartition de la base de données est déséquilibrée, cependant, vu le temps que prend l'annotation manuelle (plus de 250 images annotées), j'ai privilégié d'augmenter la part d'entraînement.

Le lien pour la base de données est le suivant : Pendule Simple

#### 2.1.2. Résultat de la détection

Après l'entraînement du modèle sur Colab Notebook, j'ai obtenu les résultats (Figure 2).

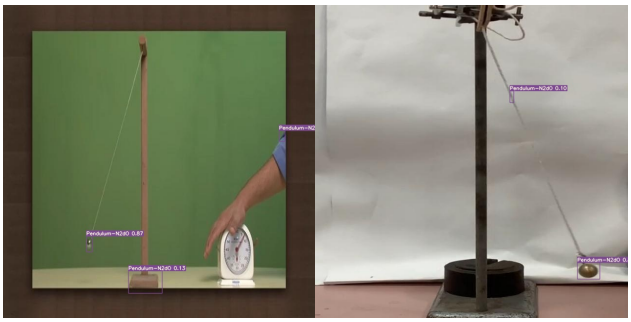

*Figure 2.* Résultat de la détection

Comme présenté dans les deux images, le modèle détecte le pendule mais aussi d'autres objets parasites sans rapport avec l'image.

Pour résoudre ce problème, nous prenons les objets avec la probabilité la plus élevée qu'il s'agisse du pendule.

Finalement, le modèle parvient à détecter avec précision le pendule 3

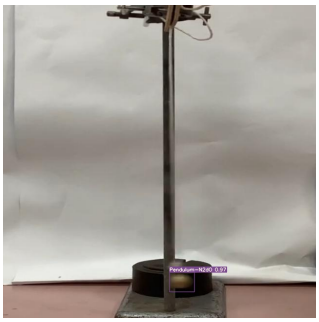

*Figure 3.* Résultat de la détection après filtrage

### 2.2. Méthode d'évaluation

Pour comparer les trois modèles, nous utiliserons deux métriques : la première est l'évolution de MSE (Mean Squared Error) au cours du temps, et la seconde est l'évolution de l'énergie du système au cours du temps.

La formule de MSE est définie comme suit :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

avec : $\hat{y}_i$ la valeur de la position prédite par le modèle, et $y_i$ la valeur de la position exacte issue de la résolution de l'ODE.
Les résultats de l'intégration pour chaque modèle sont récupérés et stockés.
Pour chaque modèle, la MSE est calculée en comparant les prédictions du modèle aux valeurs réelles.

### 2.3. Résultats d'entraînement sur des données utilisées dans le papier

#### 2.3.1. Cas du HNN, D-HNN et du MLP

Dans les trois tâches, les modèles sont entraînés avec un taux d'apprentissage de $10^{-3}$ en utilisant l'optimiseur Adam. Étant donné que les ensembles de formation étaient petits, la taille des lots était égale au nombre total d'exemples. Pour chaque ensemble de données, nous avons entraîné deux réseaux de neurones entièrement connectés : le premier était un modèle de référence qui, étant donné un vecteur d'entrée (q, p), produisait directement le vecteur $(\frac{\partial q}{\partial t}, \frac{\partial p}{\partial t})$. Le second était un HNN (Hamiltonian Neural Network) qui estimait le même vecteur en utilisant la dérivée d'une quantité scalaire en utilisant les équations de Hamilton. Lorsque cela était possible, les dérivées temporelles analytiques étaient utilisées. Sinon, le calcul était fait en utilisant la méthode des différences finies. Tous nos modèles comportaient trois couches, 200 unités cachées et des fonctions d'activation tanh. Nous les avons entraînés pendant 2000 étapes de gradient et les avons évalués sur l'ensemble de test.

Le fichier contient 555 points qui seront répartis entre 444 points d'entraînement et 111 points de test. Les données réelles de $\dot{\theta}$ en fonction de $\theta$ sont utilisées.

Les données utilisées cette fois-ci sont issues des résultats d'un papier de recherche "Distilling Free-Form Natural Laws from Experimental Data." Ces données sont très précises.(figure 4).

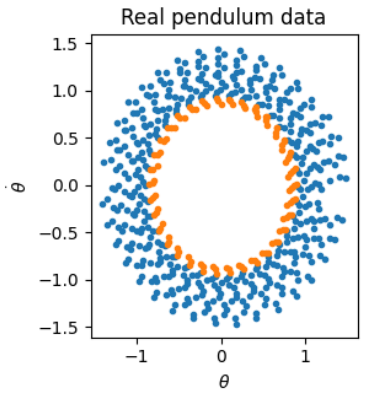


*Figure 4.* Diagramme de phase

Nous traçons également la variation de $\theta$ en fonction du temps.(figure 5)

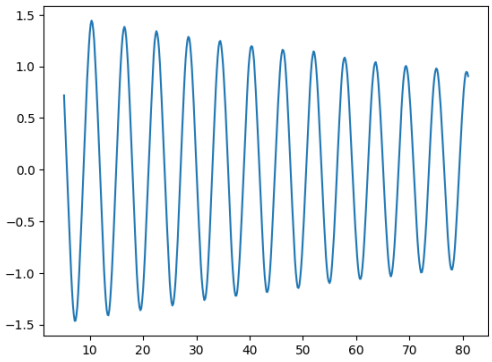

*Figure 5.* Variation de $\theta$ en fonction du temps

L'entraînement du modèle est effectué en 4800 étapes. Après avoir testé le modèle, les valeurs finales de la fonction objectif sur les données d'entraînement et de test sont présentées dans le tableau 2 :

| Modèle | Perte d'entraînement | Perte de test |
|---|---|---|
| D-HNN | 8.52e-4 | 8.37e-4 |
| HNN | 1.09e-4 | 8.71e-4 |
| MLP | 4.73e-5 | 8.33e-4 |

*Table 1.* Pertes d'entraînement et de test pour différents modèles

À noter que dans le premier et dernier graphique de la figure 6, la couleur indique l'évolution dans le temps, commençant en bleu et se terminant en rouge.

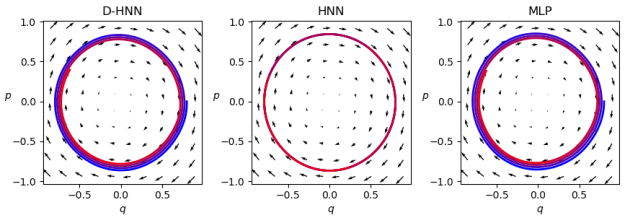


*Figure 6.* Résultats de test des trois modèles

Comme prévu, le D-HNN parvient à repérer la composante dissipative et conservative du modèle. D'autre part, le HNN force le système à être conservatif, ce qui se traduit par une forme circulaire. Enfin, le MLP diverge et n'arrive pas à reproduire fidèlement la nature physique du pendule simple.

Ces interprétations sont confirmées par les courbes de la figure 7

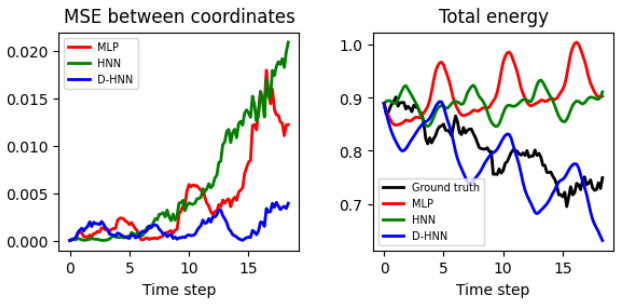


*Figure 7.* Évolution de la MSE et de l'énergie totale en fonction des pas

En effet, on remarque que la MSE reste très faible dans le cas du D-HNN. Alors que dans le cas du HNN et du MLP, elle ne cesse de croître, ce qui signifie que ces deux derniers modèles n'arrivent pas à bien représenter la valeur réelle.

Le graphique de l'énergie totale pour le pendule réel montre un autre motif intéressant.

Alors que les données réelles ne conservent pas tout à fait l'énergie totale, le HNN conserve approximativement cette quantité. En fait, il s'agit d'une limitation fondamentale des HNN : ils supposent qu'une quantité conservée existe et sont donc incapables de prendre en compte les éléments qui violent cette hypothèse, comme la friction. Pour tenir compte de la friction, nous devrions la modéliser séparément du HNN.

D'autre part, on remarque que $E_{DHNN}$ a une tendance décroissante que l'on retrouve également dans le cas réel (ground truth). Cependant, l'énergie totale du MLP a une tendance à diverger, ce qui confirme encore qu'un simple MLP ne peut pas résoudre des problématiques de la physique.

#### 2.3.2. Cas du SSGP

Pour évaluer la performance du modèle, 25 trajectoires ont été générées. L'erreur cumulée des trajectoires à un instant t est calculée en utilisant la formule :

$$\sum_{j=1}^{j_t} \left( \frac{1}{I^{test}} \sum_{i=1}^{I^{test}} \parallel \hat{x}_{i,j} - x_{i,j}^{true} \parallel \right)$$

La même formule a été utilisée pour calculer l'erreur cumulée des énergies.

Les données utilisées pour entraîner le modèle sont générées à partir de la résolution exacte des solutions de l'ODE.

### 2.4. Résultats d'entraînement sur des données tirées de la vidéo

Les données réelles de $\dot{\theta}$ en fonction de $\theta$ sont tirées de la vidéo.(8)

Les résultas plus détailler se trouvent dans le papier (10) et montrent que le modèle est meilleur que le D-HNN et HNN dans plusieurs métriques.

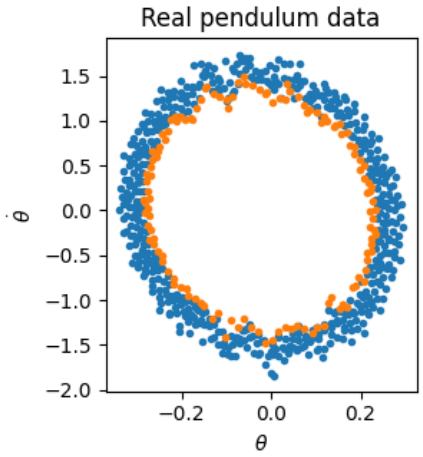


*Figure 8.* Diagramme de phase

Même avec l'utilisation du modèle YoloV7, les valeurs sont assez bruitées en raison de la faible résolution de la vidéo.

Maintenant, nous appliquons ce modèle aux données que nous avons récupérées à partir de la vidéo.

Après l'entraînement du modèle, la valeur finale de la fonction objectif est représentée dans le Tableau 2.

| Modèle | Perte d'entraînement | Perte de test |
|---|---|---|
| D-HNN | 9.23e-3 | 1.61e+0 |
| HNN | 8.78e-02 | 2.01e+00 |
| MLP | 4.26e-2 | 1.96e+00 |

*Table 2.* Pertes d'entraînement et de test pour différents modèles

Après avoir appliqué notre modèle aux données récupérées à partir de la vidéo, les résultats du D-HNN, du HNN et du MLP dans la Figure 9 ne sont pas satisfaisants.

En effet, seul le HNN parvient à récupérer la composante conservative du système, tandis que le MLP et le D-HNN échouent.

L'analyse de la courbe de MSE de la figure 10 montre qu'il n'y a pas de grande différence entre le HNN et le MLP, alors que le D-HNN a tendance à diverger.

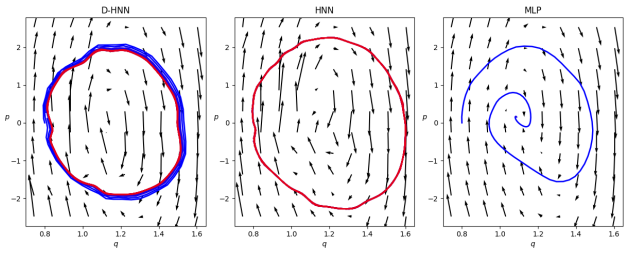


*Figure 9.* Résultats de test des trois modèles

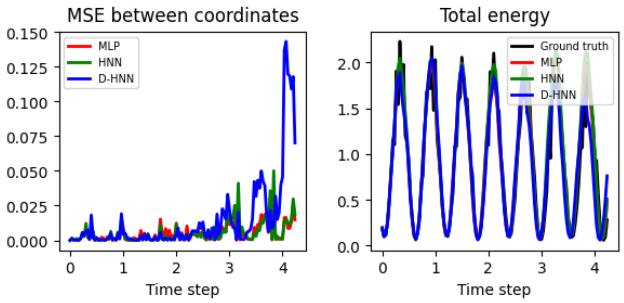


*Figure 10.* Évolution de la MSE et de l'énergie totale en fonction des pas

Dans le graphique représentant l'évolution temporelle de l'énergie totale du système, on remarque que les trois courbes oscillent de la même façon entre des valeurs minimales et maximales.

Il est clair que les trois modèles n'arrivent pas à reproduire la physique derrière ces données. Cela est principalement dû au bruit présent dans notre base de données. En effet, après séparation de la vidéo en plusieurs images, on remarque la dilatation du pendule qui occupe un plus grand espace en raison de la faible résolution de la vidéo.

### 2.5. SSGP

Le modèle est entraîné en utilisant l'optimiseur Adam avec un taux d'apprentissage de $10^{-3}$ pendant 2800 époques, implémenté dans PyTorch. L'intégration numérique est faite par la méthode adaptative de Dormand-Prince avec des tolérances relatives et absolues de $10^{-8}$. Nous avons fixé le nombre d'échantillons de Monte Carlo à K = 1 et L = 100.

Dans cette partie, j'ai essayé de conserver les données d'entraînement générées en utilisant les formules mathématiques exactes et de tester sur les données récupérées à partir de la vidéo YouTube.

Le modèle parvient à récupérer la composante conservative (figure 11)même si cette dernière n'est pas un cercle parfait, mais c'est nettement meilleur que les précédentes.

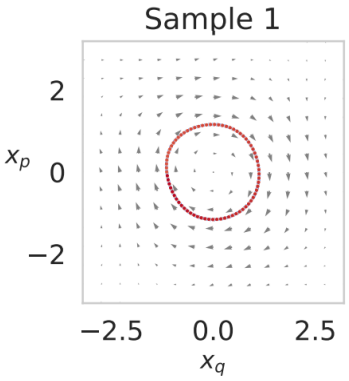


*Figure 11.* Composante conservative des données

Le modèle parvient également à bien reproduire la composante dissipative (figure 12)

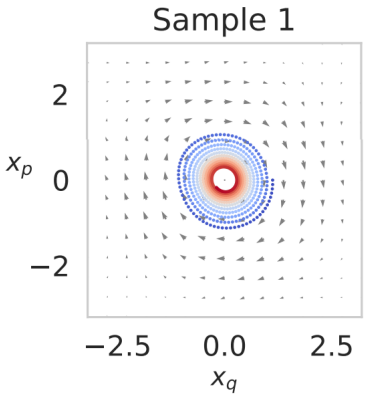


*Figure 12.* Composante dissipative des données

En effet, le modèle a une tendance convergente vers le point (0,0) qui représente la valeur finale du mouvement du pendule dans le cas réel.

Après l'entraînement du modèle, nous pouvons récupérer la valeur de la fonction coût (figure 13).

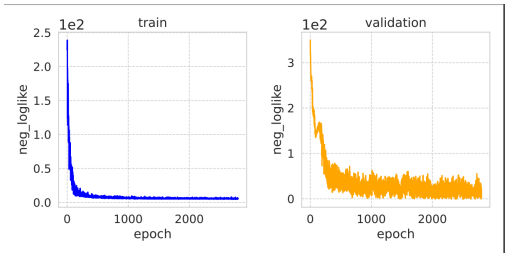


*Figure 13.* Évolution de la fonction coût en fonction des pas d'entraînement

Ainsi, le modèle SSGP parvient à bien reproduire la réalité

physique, mais il faut mettre en perspective l'entraînement du modèle sur des données exactes issues de la résolution des équations numériques, chose qui n'est pas toujours possible.

Après l'entraînement du modèle, nous récupérons les différentes valeurs de la MSE en fonction du temps, la courbe de la figure 14.

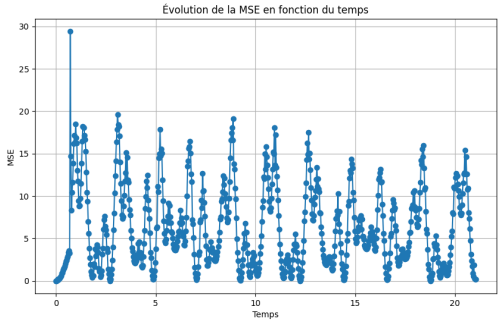


*Figure 14.* Évolution de la MSE en fonction du temps

Ce résultat n'est pas du tout souhaitable, en effet, la MSE devrait diminuer en fonction du temps.

Nous gardons également de manière continue les différentes valeurs de l'énergie totale du système (figure 15).

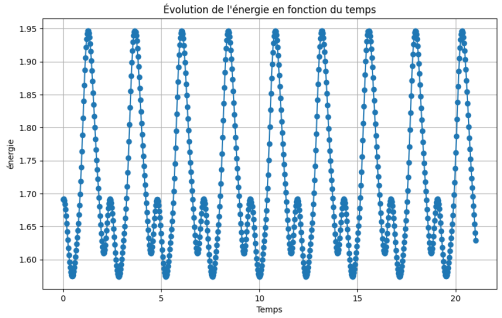


*Figure 15.* Évolution de l'énergie totale en fonction du temps

Ce résultat n'est pas non plus souhaitable, en effet, l'énergie totale du système devrait avoir une tendance décroissante.

### 2.6. Récapitulatif décisif

Afin de comparer les modèles de façon plus concise, nous avons imaginé un indicateur final calculé à partir de la MSE divisée par la durée de l'entraînement :

$$\text{Indicateur} = \frac{\text{MSE}}{T_{\text{entraînement}}}$$

Nous calculons la valeur de cet indicateur pour différents modèles. Les résultats sont regroupés dans le tableau 3

| Modèle | MSE |
|---|---|
| MLP | $4.35 \times 10^{-3}$ |
| HNN | $5.86 \times 10^{-3}$ |
| D-HNN | $1.30 \times 10^{-3}$ |
| SSGP | $6.3 \times 10^{0}$ |

*Table 3.* Valeurs de l'indicateur de la MSE

Dans le cas de l'utilisation des données de la vidéo par la méthode SSGP, le calcul de la MSE n'est pas approprié car nous ne connaissons pas les valeurs exactes de la position du pendule pour pouvoir comparer.

Ce n'est pas le cas pour les D-HNN et HNN, car nous n'avons pas supposé la présence de bruit dans les données.

Une solution envisageable serait de disposer des données du pendule (longueur du fil, masse, vitesse initiale) afin de résoudre l'équation numérique par la suite. Cependant, cette solution n'est pas applicable car l'objectif principal était d'apprendre le mouvement du pendule à partir d'une vidéo.

Nous effectuons maintenant la même analyse mais en remplaçant la MSE par l'énergie du système. Les résultats sont regroupés dans le tableau 4

En ce qui concerne l'énergie totale du système, nous remarquons qu'elle reste proche de la valeur réelle, mais pas parfaitement exacte.

Avec le critère conçu, nous ne pouvons pas comparer toutes les méthodes. Cependant, en nous référant aux courbes représentant les différents champs de vecteurs conservatifs et dissipatifs, nous pouvons clairement valider l'utilité du modèle SSGP.

| Modèle | Énergie |
|---|---|
| Énergie Réelle | $8.03 \times 10^{-1}$ |
| MLP | $9.07 \times 10^{-1}$ |
| HNN | $8.88 \times 10^{-1}$ |
| D-HNN | $7.78 \times 10^{-1}$ |
| SSGP | $1.7 \times 10^{0}$ |

*Table 4.* Valeurs de l'indicateur de l'énergie

## 3. Conclusion

Dans ce travail, nous avons exploré les capacités et les limitations de divers modèles de réseaux neuronaux, en nous concentrant particulièrement sur les Réseaux Neuronaux Hamiltoniens (HNNs), les Réseaux Neuronaux Hamiltoniens Dissipatifs (D-HNNs) et les MLPs (Multilayer Perceptrons). L'analyse comparative, à travers des métriques telles que l'erreur quadratique moyenne (MSE) et les valeurs d'énergie normalisée, a mis en évidence l'efficacité de ces modèles

dans l'apprentissage et la prédiction des dynamiques des systèmes physiques à partir des données.

Cette étude souligne le potentiel de l'incorporation des principes physiques dans les architectures de réseaux neuronaux pour améliorer leur efficacité d'apprentissage et leur précision prédictive, en particulier dans le contexte de la modélisation des systèmes physiques complexes avec un bruit inhérent et des données limitées.

Une des améliorations envisageables est de remplacer les MLP par des KAN (Kolmogorov-Arnold Networks) (7). Ces derniers exhibent certaines caractéristiques particulières, surtout en ce qui concerne les PINNs (Physics-Informed Neural Networks). Ils possèdent la capacité d'interpréter les résultats et de remonter plus facilement aux lois de la physique.

Le répertoire GitHub où l'on peut trouver la majorité des codes que j'ai utilisés. : Lien.

## References


[1] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002, 2019.

[2] Wikipedia contributors. Théorème d'approximation universelle. https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_d%27approximation_universelle, 2024. Accessed: 2024-05-28.

[3] D. A. Garanin. Hamiltonian mechanics. https://www.lehman.edu/faculty/dgaranin/Mechanics/Hamiltonian_mechanics.pdf, n.d. Accessed: 2024-05-28.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[5] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[6] Simon Haykin. *Neural Networks and Learning Machines*. Pearson Education, 2009.

[7] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024.

[8] Andrew Sosanya and Sam Greydanus. Dissipative hamiltonian neural networks: Learning dissipative and conservative dynamics separately. *arXiv preprint arXiv:2201.10085*, 2022.

[9] Yusuke Tanaka, Tomoharu Iwata, et al. Symplectic spectrum gaussian processes: learning hamiltonians from noisy and sparse data. *Advances in Neural Information Processing Systems*, 35:20795–20808, 2022.

[10] Michalis K Titsias and Neil D Lawrence. Bayesian gaussian process latent variable model. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 844–851, 2009.

[11] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023.

[12] Y. D. Zhong, B. Dey, and A. Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2020.


(11) (8) (9) (7) (1) (10) (5) (12) (2) (3) (4)