Problèmes spectraux en mécanique quantique avec réseaux de neurones - rapport de stage

ÉTUDIANT : CLÉMENT LOTTEAU, ENCADRANTS : JÉRÔME MARGUERON ET HUBERT HANSEN

Dans ce rapport, nous étudions dans un premier temps la capacité des réseaux de neurones artificiels à approximer des fonctions continues à support compact. Nous décrivons ensuite un programme original de minimisation de l'énergie d'une particule dans un puits de potentiel reposant sur cette capacité d'approximation qu'ont les réseaux.

TABLE DES MATIÈRES

1. INTRODUCTION

Le calcul des états et des énergies propres pour un potentiel quelconque est un problème général en mécanique quantique qui s'étend de la physique atomique à l'étude des quarks. On trouve ces états et leurs énergies par la résolution de l'équation de Schrödinger indépendante du temps mais l'absence de solution analytique pour la plupart des potentiels nous contraint à recourir à des solveurs numériques. Ce rapport présente les résultats de deux solveurs développés en autonomie : un algorithme de diffusion Runge-Kutta d'ordre 4 et une méthode de minimisation stochastique originale s'inscrivant dans le cadre des méthodes variationnelles et reposant sur la qualité d'approximateurs universels des réseaux de neurones. L'idée d'utiliser les réseaux de neurones comme solveurs de l'équation de Schrödinger indépendante du temps nous est venu de l'article *Machine learning the deuteron* écrit par JWT. Keeble et A. Rios. [3]

Afin de trouver les états et les énergies propres d'un potentiel $V(x)$, on doit résoudre l'équation de Schrödinger indépendante du temps (à une dimension ici) où m est la masse d'une particule :

$$\left[\frac{-\hbar^2}{2m}\frac{d^2}{dx^2} + V(x)\right]\psi(x) = E\psi(x) \tag{1}$$

Si $\psi(x)$ est un état propre, son énergie peut être obtenue en multipliant (1) à gauche par $\psi(x)$, puis en intégrant l'énergie cinétique par partie :

$$E = \frac{\frac{-\hbar^2}{2m}\left(\left[\psi\psi'\right]_{-\infty}^{+\infty} - \int_{-\infty}^{+\infty}|\psi'|^2 dx\right) + \int_{-\infty}^{+\infty} V(x)|\psi|^2 dx}{\int_{-\infty}^{+\infty}|\psi|^2 dx} \tag{2}$$

Le potentiel test que nous étudierons ici est celui d'un oscillateur harmonique à une dimension ($V(x) = \frac{1}{2}m\omega x^2$) dont les énergies propres analytiques sont $E = \hbar\omega(\frac{1}{2} + n)$ où n est un entier naturel. Pour l'état fondamental :

$$E_c = \frac{\hbar\omega}{4} \quad ; \quad E_p = \frac{\hbar\omega}{4} \quad ; \quad E_{totale} = \frac{\hbar\omega}{2} \tag{3}$$

On prendra m, $\hbar$, c et ω égaux à 1 par la suite.

A. États propres d'un oscillateur harmonique à une dimension : méthode Runge-Kutta

n	Énergie
0	0.487663
1	1.46298
2	2.43830
3	3.41361
4	4.38893

TABLE 1. Énergies trouvées grâce à la propagation RK4

L'algorithme RK d'ordre 4 que j'ai programmé et complété d'une méthode de tir nous permet de trouver les énergies (table 1) et les fonctions d'ondes (figure 1) des états liés. On observe que, contrairement à la théorie, les énergies trouvées ne sont pas 0.5, 1.5, 2.5...etc. Une erreur moyenne de 6% est commise. L'écart entre chaque niveau est constant et vaut environ 0.97532 (contre 1 théoriquement). Ces problèmes sont peut-être dus à l'accumulation d'erreurs de calcul au fil de la propagation. Cette méthode n'étant cependant pas le sujet principal du stage, nous nous sommes concentrés sur les réseaux de neurones. Des résultats supplémentaires sont disponibles sur mon GitHub [9].

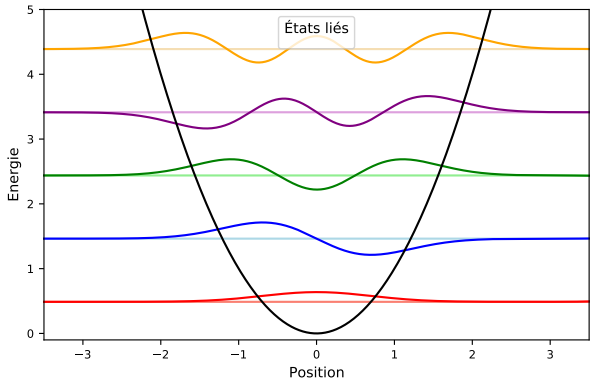


Fig. 1. États liés d'un oscillateur harmonique. Les fonctions d'onde (non normalisées) sont alignées sur leurs énergies.

2. RÉSEAUX DE NEURONES - ILLUSTRATION DU THÉORÈME D'APPROXIMATION UNIVERSELLE

Selon le théorème d'approximation universelle [1], un réseau de neurone peut reproduire n'importe quelle fonction à valeurs réelles et à support compact. Nous essayons ici d'illustrer ce théorème en approximant l'état fondamental d'un oscillateur harmonique. Des tutoriels d'initiation aux réseaux de neurones écrits par Colin Bernet sont disponibles en bibliographie : [4], [5] et [6].

A. Réseaux de neurones - concept et vocabulaire

Inspirés des neurones biologiques, les neurones artificiels composant le réseau sont tous dotés d'une *fonction d'activation* qui détermine l'activation ou la non activation des neurones pour une entrée donnée. La fonction d'activation que nous utilisons ici est une *ReLU* (*Rectified Linear Unit*) qui prend la forme suivante :

$$R(z) = \begin{cases} 0 & \text{quand} \quad z \leq 0 \\ z & \text{quand} \quad z > 0 \end{cases} \tag{4}$$

Un neurone doté d'une telle fonction d'activation est éteint lorsque $z \leq 0$ et est allumé pour $z > 0$. Cette variable prend la forme $z = ax + b$ où x est la valeur qui entre dans la fonction d'activation, et où a et b sont des quantités modifiées par entrainement du réseau, respectivement le *poids* et le *biais*. Un neurone est donc composé d'une fonction d'activation qui prend en entrée x les sorties $R(z)$ des neurones de la couche qui précède la sienne (figure 2). Prenons un exemple fictif dans lequel on cherche à ce que notre réseau reproduise la fonction discrète suivante :

$$f(x) = \begin{cases} 1 & \text{quand} \quad x = 0 \\ 7 & \text{quand} \quad x = 1 \\ 3 & \text{quand} \quad x = 2 \end{cases} \tag{5}$$

On la représente par deux listes de trois nombres : $x = [0, 1, 2]$ et $y_{cible} = [1, 7, 3]$ puis on entraine notre réseau à reproduire y_{cible}. Pour ce faire, on commence par initialiser le réseau avec des paramètres (a et b) aléatoires puis on lui fournit x et y_{cible}. On propage l'ensemble des données x dans le réseau qui se charge de faire une prédiction y_{pred} qu'on compare ensuite à y_{cible}. Un *epoch* correspond à une propagation de toutes les données x.

Entrainement :

Epoch 1 : Le réseau fait une première prédiction $y_{pred} = [32, -9, 64]$ éloignée de la liste attendue. La *fonction de coût* calcule l'écart quadratique moyen entre y_{cible} et y_{pred} appelé perte (*loss*). Le réseau calcule ensuite le gradient de la fonction de coût par rapport à a et b pour savoir comment les modifier afin de minimiser le loss. Il les modifie en conséquence, fin du premier epoch.

Epoch 2 : Le réseau prédit $y_{pred} = [11, 3, 9]$. La prédiction est meilleure grâce au premier entrainement mais doit être améliorée. Le réseau calcule le loss, le gradient, puis modifie ses paramètres et recommence le processus pour l'epoch 3. Ces calculs sont gérés automatiquement par Keras [8], la librairie Python que nous utilisons ici.

Géométries :

On voit sur la figure 2 que les réseaux peuvent être décomposés en trois parties : la couche d'entrée (rouge) depuis laquelle les données x sont propagées vers les couches cachées (violet) puis le résultat final y_{pred} est calculé en couche de sortie (bleu). Les réseaux que nous étudions ici prennent une discrétisation de l'espace (x) en entrée et renvoient une fonction d'onde (y_{pred}) en sortie. Par la suite, j'utiliserai le mot "couche" pour désigner les couches cachées des réseaux. Aussi, un réseau 200x200 par exemple désignera un réseau à deux couches cachées, toutes deux composées de 200 neurones. Tous les réseaux que nous étudions disposent d'une entrée et d'une sortie seulement.

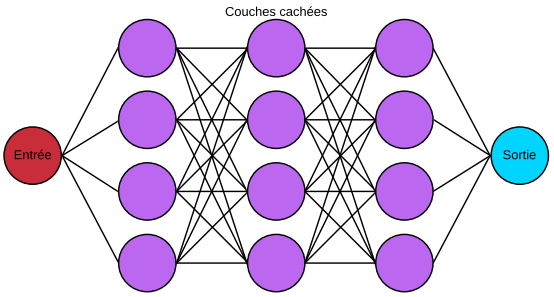


Fig. 2. Schéma d'un réseau de neurones : entrée, 3 couches cachées et sortie.

B. Caractéristiques de la machine utilisée

Tout le travail lié aux réseaux de neurones a été réalisé sur Google Colab [7] qui permet d'exécuter des notebooks Python depuis un navigateur. Les caractéristiques de la machine sont : 2 processeurs Intel(R) Xeon(R) CPU @ 2.20GHz ; 1 coeur ; 46 bits physiques ; 48 bits virtuels.

C. Reproduction de l'état fondamental d'un oscillateur harmonique

On cherche maintenant à reproduire l'état fondamental d'un oscillateur harmonique à une dimension. On dispose d'une liste x discrétisée de -5 à 5 sur 10001 points, et d'une liste y_{cible} calculée à partir de la solution analytique [2] de l'équation (1) pour l'état fondamental ψ_{EF} d'énergie E_{EF} :

$$\psi_{EF}(x) = \left(\frac{m\omega}{\pi\hbar}\right)^{\frac{1}{4}} e^{\frac{-m\omega x^2}{2\hbar}} \quad ; \quad E_{EF} = \frac{1}{2} \tag{6}$$

Les courbes noire, rouge et orange de la figure 3 illustrent une fonctionnalité de Keras permettant de mesurer le loss à chaque epoch. La courbe bleue est une moyenne sur 30 runs du programme de reproduction. Sur la courbe rouge, on voit que le minimum de la fonction de coût a été atteint vers le 45^e epoch environ. Néanmoins, le réseau continue à modifier ses paramètres lors des epochs suivants et oscille autour du minimum de la fonction de coût. Le loss augmente et les prédictions s'éloignent de la

cible. L'aléatoire devient important à partir du 30^e epoch environ. Autrement dit, au delà de ce seuil, deux runs du programme pourront donner des résultats très différents.

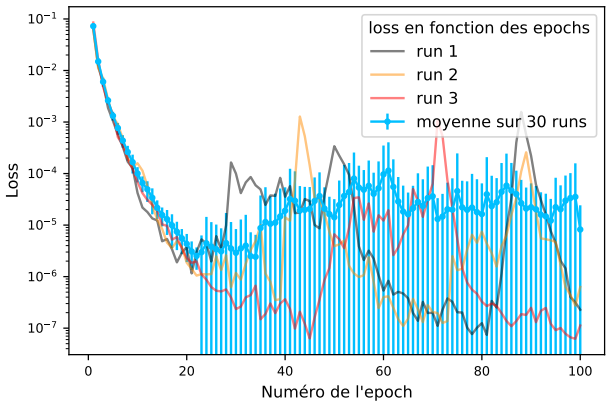


Fig. 3. Écart quadratique moyen (*loss*) entre ψ_{EF} et les prédictions du réseau (200x200x200) en fonction des epochs.

Une étude du loss en fonction des epochs a été faite pour comparer l'impact de la géométrie sur la minimisation du loss [9]. On y voit que l'ajout de non linéarité (plusieurs couches) permet d'atteindre un loss plus faible, et ce en moins d'epochs que pour un réseau à une couche. On y voit aussi que l'augmentation du nombre de paramètres permet d'obtenir des loss plus bas mais que les oscillations autour du minimum de la fonction de coût ont une plus grande amplitude qu'avec des réseaux moins denses.

D. Calcul de l'énergie de la reproduction

Une fois l'état fondamental reproduit par le réseau, on peut en extraire l'énergie avec l'équation (2). Une première tentative d'extraction nous avait posé problème du fait que les prédictions du réseau sont des morceaux de droites. Un bon apprentissage permet de lisser le résultat mais la dérivée de la prédiction restait discontinue. Nous avons donc extrait l'énergie à partir de splines cubiques de la prédiction dont le calcul [11], la dérivation et l'intégration [10] sont gérés par la librairie SciPy. La figure 4 présente les résultats du calcul des énergies et de la norme de 50 prédictions effectuées par le réseau après entrainement. On observe que l'énergie totale des prédictions est toujours supérieure à l'énergie théorique de 0.5, celle-ci étant le minimum atteignable, ce qui illustre bien qu'on trouve l'état fondamental de façon variationnelle.

3. MÉTHODE VARIATIONNELLE ET MINIMISATION DE L'ÉNERGIE : UTILISATION STOCHASTIQUE DES RÉSEAUX DE NEURONES

Jusqu'ici nous avons étudié la qualité d'approximateur universel des réseaux de neurones mais notre problème n'est pas résolu. Notre but maintenant est de développer une méthode pour que le programme trouve l'état fondamental sans le connaitre au préalable. Pour résoudre ce problème, j'ai eu l'idée du programme détaillé en figure 6 dans lequel le réseau cherche dans un premier temps à reproduire une fonction d'onde cible constante. Le réseau fait un epoch pour s'en approcher, puis une prédiction, et si l'énergie de la prédiction est inférieure à celle de sa cible,

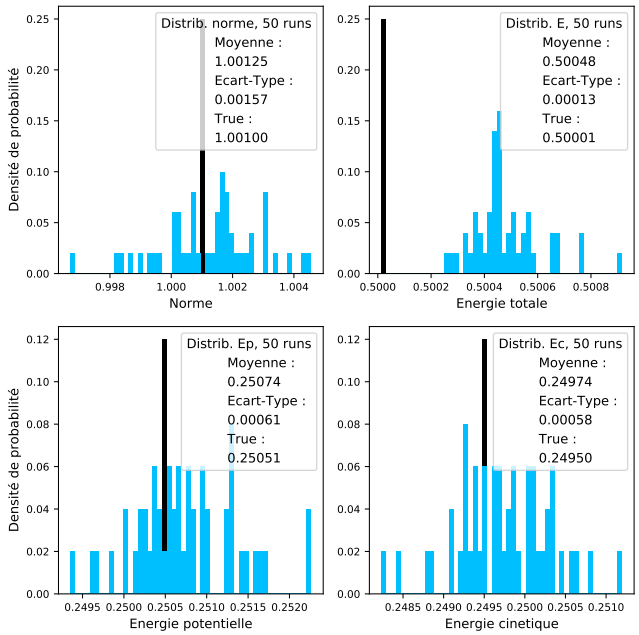


Fig. 4. Distribution des énergies et des normes de 50 prédictions du réseau. Les barres noires sont des repères visuels de la fonction à fitter.

la prédiction devient la nouvelle cible. La méthode utilisée est donc stochastique, on se sert des petites fluctuations (figure 3) pour nous rapprocher de l'état d'énergie minimale.

A. Première version du programme

Dans la première version du programme, la première cible était une fonction composée de nombres aléatoires entre 0 et 1 et l'étape 6 n'existait pas. De plus, je détruisais le réseau puis le reconstruisais à l'étape 8 en espérant générer un grand loss et ainsi accélérer le processus de minimisation. Cette méthode a donné le résultat visible sur la figure 5. Les prédictions tendaient

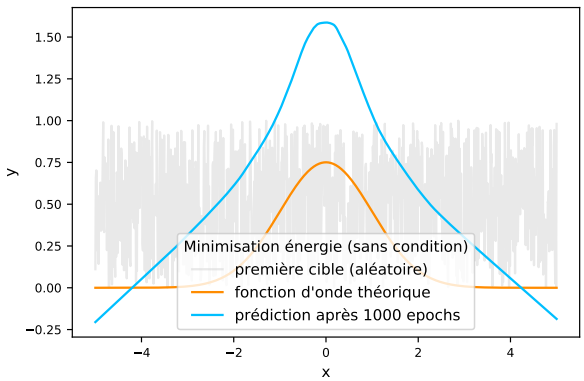


Fig. 5. Première version du programme. 1000 itérations, discrétisation sur 1000 points.

vers une gaussienne au bout de 1000 itérations mais le processus était très lent et les résultats trop éloignés de la théorie. J'ai donc ajouté 3 conditions (figure 6, étape 6) : la fonction d'onde doit être positive, symétrique et normée. Les deux premières conditions

donnaient de meilleurs résultats que la figure 5 mais la précision restait faible. C'est la normalisation qui a permis d'atteindre des résultats similaires à la figure 7. La déconstruction-reconstruction du réseau avec des paramètres aléatoires avait permis d'obtenir les résultats de la figure 5, mais elle faisait perdre beaucoup de temps au programme lorsque les prédictions était déjà bonnes. Il ne restait plus qu'à affiner le résultat mais le programme continuait de reprendre son apprentissage à 0 à chaque fois qu'une meilleure cible était trouvée. En regardant la figure 3, on voit que le loss est grand au début de l'entrainement. Cela signifie que les prédictions du réseau sont très éloignées de la cible alors qu'on cherche de petites variations pour affiner le résultat. De plus, les nouvelles conditions permettaient déjà d'orienter le programme vers une gaussienne.

B. Dernière version du programme

Dans cette version, j'ai arrêté la déconstruction-reconstruction du réseau et ai aussi changé la discrétisation en passant de 1000 à 100 points pour diminuer le temps de calcul. La figure 6 présente la dernière version en date du programme dont le code commenté en anglais est disponible sur mon GitHub [9].

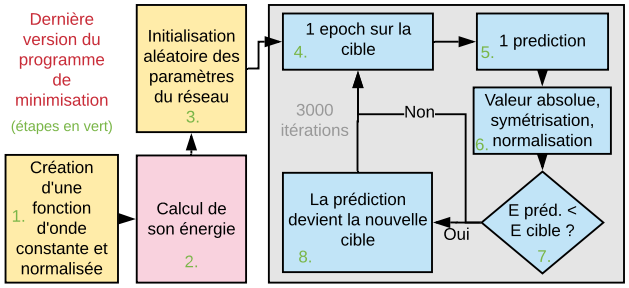
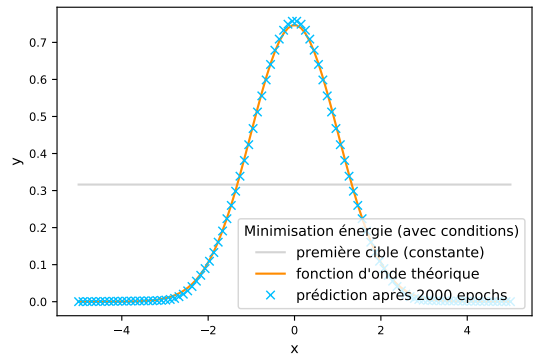


Fig. 6. La première cible est une fonction constante. On garde la prédiction avec l'énergie la plus faible en sortie de boucle.

La figure 7 illustre un résultat moyen de l'algorithme après 2000 itérations. On peut y voir que l'écart entre la fonction d'onde théorique et le résultat de l'algorithme semble être le plus important là où la courbure est forte. On voit que la partie linéaire du résultat (en bas à gauche) est inférieure à la théorie alors que le sommet est supérieur. D'autres résultats semblaient comporter les mêmes problèmes. Le but du réseau étant de "se reproduire lui-même", il est possible qu'une droite soit pour lui plus simple à reproduire qu'une courbe. Cela rendrait le réseau "rigide", réticent à changer certains de ses paramètres qui permettraient de gagner en précision sur l'énergie, et éventuellement en temps de calcul. L'échelle logarithmique illustre la difficulté du programme à produire des résultats qui convergent vers 0 à l'infini.

Pour étudier la convergence du programme, on calcule l'écart entre l'énergie trouvée par le programme et l'énergie théorique (équation (6). On trace ensuite cet écart en fonction du temps de calcul CPU pour observer la convergence. Le résultat est illustré par la figure 9 page 5 sur laquelle on a fait la moyenne des écarts et du temps CPU sur 30 runs du programme. On y voit notamment que la convergence dépend beaucoup du nombre de paramètres du réseau et de la géométrie choisie. Par exemple, les réseaux à une couche et les réseaux à 500 paramètres ne convergent pas. On remarque aussi que, pour les réseaux à 6 couches, 5000 paramètres semble être un meilleur choix que 30600, contrairement aux réseaux à 3 couches. La figure 8 représente le temps de calcul CPU en fonction de la géométrie



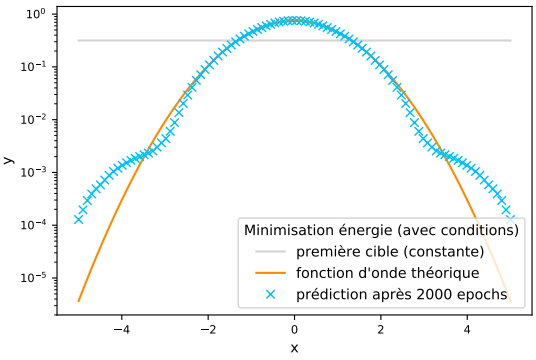


Fig. 7. Résultat moyen de la dernière version du programme en échelle linéaire puis logarithmique. 2000 itérations, discrétisation sur 100 points.

utilisée. On y voit que les réseaux à 500 et 5000 paramètres ont des temps CPU similaires pour la plupart des géométries. 30600 est toujours un peu au dessus et même très au dessus pour le réseau à 1 couche.

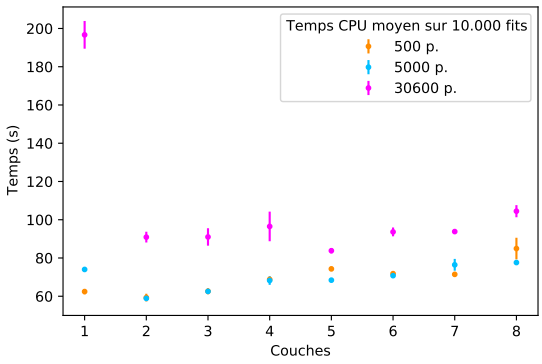


Fig. 8. Temps CPU au bout de 10000 itérations, moyenné sur 30 runs.

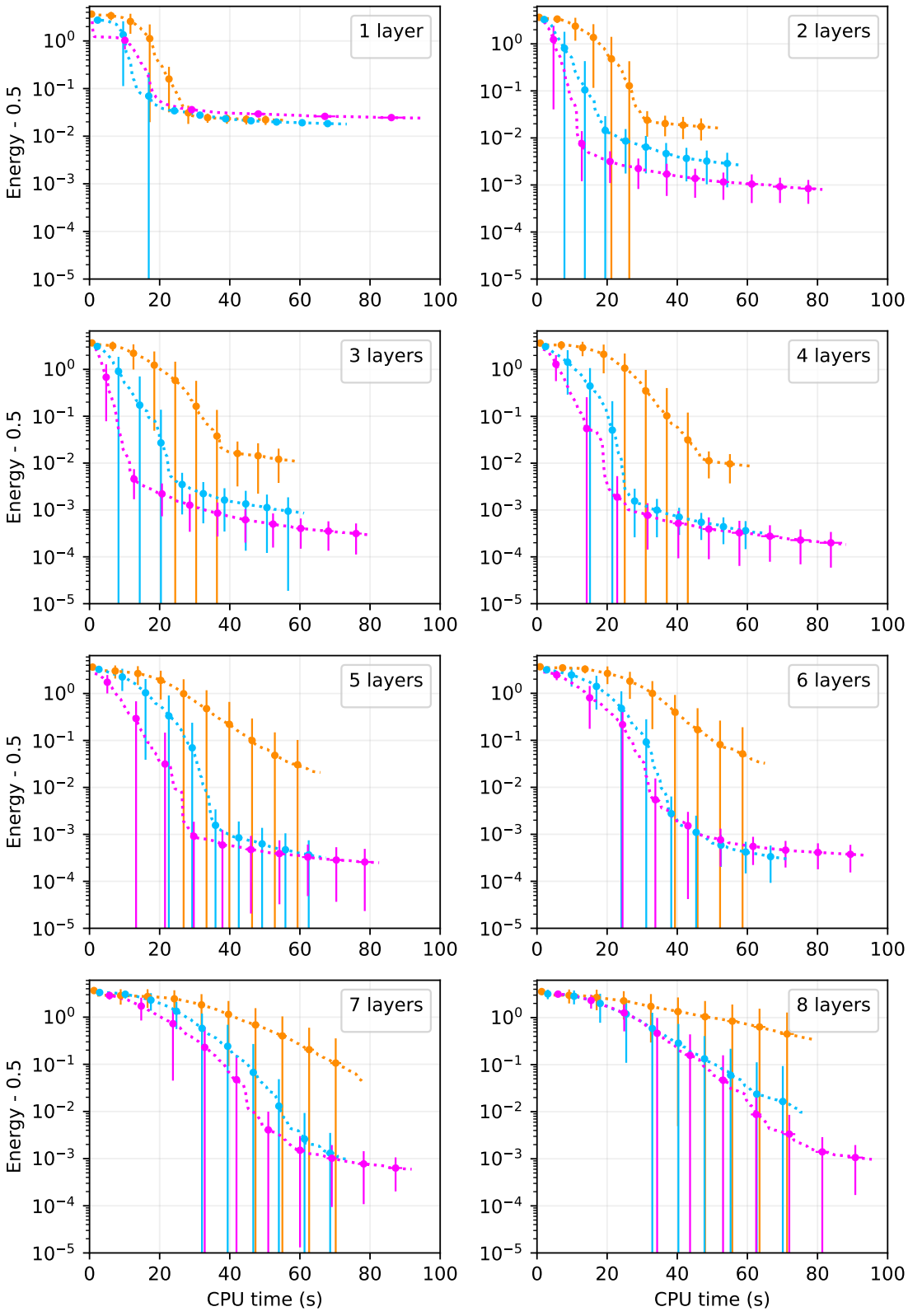


Fig. 9. Convergence du programme de minimisation. Orange : 500 paramètres ; Bleu : 5000 ; Rose : 30600. Les courbes s'arrêtent lorsque le programme a effectué 10000 itérations, à l'exception de "1 couche - rose" qui est coupée à 5000 itérations pour des raisons de lisibilité (200 secondes au total). La précision et le temps sont moyennés sur 30 runs.

On confirme aussi que le programme fonctionne pour des potentiels non harmonique avec la figure 10 sur laquelle les potentiels étudiés sont respectivement :

$$V(x) = |x|$$
$$V(x) = \frac{1}{2}x^2 + 3e^{-4x^2} \quad (7)$$

Une fonction constante étant très éloignée du résultat final, j'ai utilisé ici l'état fondamental de l'oscillateur harmonique comme première cible pour accélérer le processus de minimisation. On observe sur la figure 10 que l'état fondamental pour $|x|$ tend moins vite vers 0 que pour x^2. Ce résultat est cohérent avec la croissance plus rapide du potentiel harmonique devant celle de $|x|$ lorsque $x > 1$. Inversement, lorsque $x < 1$, x^2 tend plus vite vers 0 que $|x|$ et on observe que l'état propre de $|x|$ est supérieur à celui de x^2 dans cette zone.

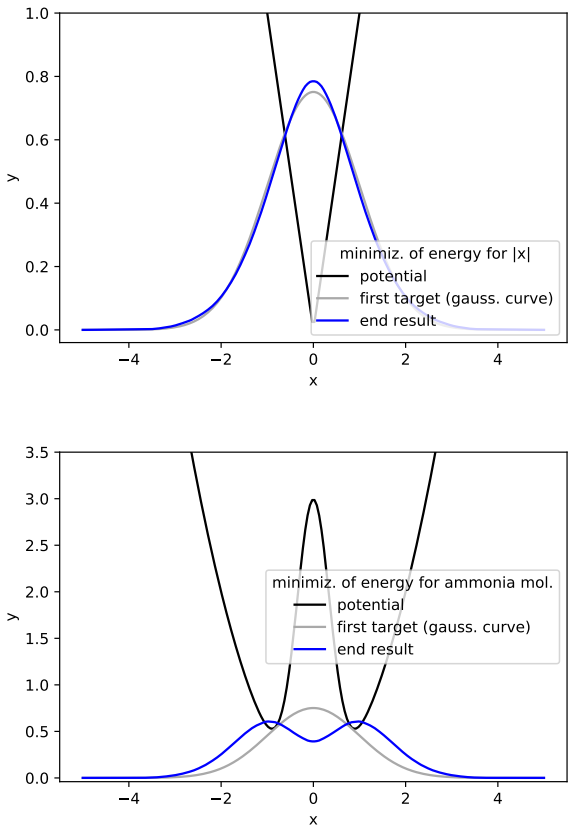


Fig. 10. Minimisationde l'énergie pour les potentiels de l'équation (7) Discrétisation de l'espace sur 200 points. 7000 itérations.

4. CONCLUSION

Nous avons étudié deux méthodes numériques de résolution de l'équation de Schrödinger indépendante du temps permettant de trouver la fonction d'onde et l'énergie de l'état fondamental d'un oscillateur harmonique. Contrairement à un algorithme de diffusion comme RK4, le programme de minimisation stochastique ne permet de trouver que l'état fondamental pour le moment et il est encore tôt pour conclure sur les performances du programme stochastique du fait que les résultats de la figure 9 en annexe ne sont pas complètement convergés après 10000 itérations pour la plupart des géométries étudiées. On sait néanmoins que les réseaux de neurones peuvent être utilisés dans le cadre des méthodes variationnelles et de nouvelles questions se posent déjà : y a-t-il un nombre optimal de points pour la discrétisation de l'espace ? Quelles sont ses performances sur des temps plus longs ? Que peuvent apporter de nouvelles géométries ? Geler des couches dans la limite asymptotique permet-il d'accélérer la minimisation ? Comment se comporte le loss lorsqu'on change la cible en court d'apprentissage ? Quelles sont les performances du programme à deux et trois dimensions ? Nous réfléchissons aussi à une méthode permettant de trouver les états excités, éventuellement en travaillant dans un sous-espace de fonctions orthogonales.

BIBLIOGRAPHIE

[1] Balázs Csanád CSÁJI. "Approximation with Artificial Neural Networks, MSc Thesis". In : *Eötvös Loránd University (ELTE), Budapest, Hungary* 24 :48 (2001). URL : https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.2647&rep=rep1&type=pdf.

[2] Claude COHEN-TANNOUDJI, Bernard DIU et Franck LALOË. *Mécanique quantique Tome 1*. EDP Sciences, 2018, p. 371-378. ISBN : 9782759822874.

[3] J. W. T. KEEBLE et A. RIOS. "Machine learning the deuteron". In : (2019). URL : arXiv:1911.13092.

[4] Colin BERNET. *Handwritten Digit Recognition with scikit-learn*. URL : https://thedatafrog.com/en/articles/handwritten-digit-recognition-scikit-learn/.

[5] Colin BERNET. *Le réseau à un neurone : régression logistique*. URL : https://thedatafrog.com/fr/articles/logistic-regression/.

[6] Colin BERNET. *Premier réseau de neurones avec keras*. URL : https://thedatafrog.com/fr/articles/first-neural-network-keras/.

[7] GOOGLE. *Colaboratory*. URL : https://colab.research.google.com/.

[8] KERAS. *Model training*. URL : https://keras.io/api/models/model_training_apis/.

[9] Clément LOTTEAU. *Mon GitHub*. URL : https://github.com/quadrivecteur?tab=repositories.

[10] SCIPY. *Integration avec SciPy*. URL : https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html.

[11] SCIPY. *Interpolation avec SciPy*. URL : https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html.