Dédicace

À

mon père **Komlavi BASSA**,
ma mère **Ablavi EDOULI**,
mes **frères** et **sœurs**.

Remerciements

Je passe par ces quelques mots pour exprimer mes profonds remerciements à tous ceux qui ont contribué à l'aboutissement de ce mémoire. J'exprime ma gratuite à Monsieur **Frédéric HOLWECK**, d'abord pour l'opportunité qu'il m'a offerte en m'acceptant sous sa direction, et en suite pour son accompagnement tout au long du stage. Ses remarques et ses conseils m'ont été d'une grande aide et m'ont permis d'apprendre sur le monde de la recherche.

Je tiens à remercier les dirigeants de ColibriTD ; **Dr Laurent GUIRAUD** et **Hacène GOUDJIL** de m'avoir accepté au sein de leur entreprise. Je les remercie aussi pour les conditions qui m'ont été offertes pour produire ce travail.

Mes remerciements vont aussi à l'endroit des chercheurs de ColibriTD, spécialement **Youcef MOHDEB**, **Hugo BARTOLOMEI** et **Henri de BOUTRAY** pour leurs aides et apports tout au long de mon stage.

Enfin, je tiens à remercier les responsables du master, les enseignants et mes camarades pour leurs aides et écoutes tout au long de ces deux années passées à l'EPL.

0.1 Résumé

Les phénomènes physiques peuvent être modélisés sous forme d'équation différentielle. Ainsi, la résolution des équations différentielles est essentielle à la compréhension de ces phénomènes. Face à l'incapacité de résoudre certains problèmes de manière analytique, les méthodes numériques ont vu le jour. Mais très vite elles se heurtent aussi aux limites matérielles de l'informatique classique et sont inefficaces devant certains problèmes complexes. La naissance des domaines comme l'informatique quantique et les réseaux de neurones, ouvre de nouveaux horizons avec de nouvelles méthodes.

Dans ce mémoire, nous étudions deux méthodes de résolution d'équation différentielle que sont les physics-informed neural networks et l'hybrid differential equations solver (développé par l'entreprise colibriTD). Premièrement, nous rappelons le fonctionnement des réseaux de neurones. Deuxièmement, nous introduisons les principes de l'informatique quantique, puis nous expliquons le fonctionnement de l'algorithme variationnel quantique. Troisièmement, nous parlons des physics-informed neural networks. Quatrièmement, nous étudions l'hybrid differential equations solver. Enfin, nous appliquons ces méthodes à la résolution de l'équation de la flamme laminaire unidimensionnelle, et effectuons une comparaison des deux méthodes dans ce cas spécifique. Cette comparaison montre que le HDES (hybrid differential equations solver) nécessite moins de paramètres et moins d'itérations, et que les mêmes optimiseurs n'ont pas les mêmes performances avec les deux méthodes.

Key words : Réseau de neurones, Physics-informed neural networks (PINNs), Informatique quantique, Algorithme variationnel quantique, Informatique quantique, hybrid differential equations solver.

0.2 Abstract

Physical phenomena can be modeled in the form of differential equations. Solving differential equations is therefore essential to understanding these phenomena. Faced with the inability to solve certain problems analytically, numerical methods were born. But they soon came up against the hardware limitations of conventional computing and were ineffective for certain complex problems. The emergence of fields such as quantum computing and neural networks is opening up new horizons with new methods.

In this thesis, we study two methods for solving differential equations : physics informed neural networks and the hybrid differential equations solver (developed by the company colibriTD). First, we review how neural networks work. Secondly, we introduce the principles of quantum computing, then explain how the quantum variational algorithm

works. Thirdly, we talk about physics-informed neural networks. Fourthly, we study the hybrid differential equation solver. We highlight some points of comparison between these two methods. Finally, we apply these methods to the solution of the one-dimensional laminar flame equation, and carry out a comparison of the two methods in this specific case. This comparison shows that the HDES (hybrid differential equations solver) requires fewer parameters and fewer iterations, and that the same optimizers do not perform equally well with the two methods.

Key words : Neural network, Physics-informed neural networks (PINNs), Quantum computing, Variational quantum algorithm, Quantum computing, Hybrid differential equation solver.

Table des matières

0.1 Résumé I
0.2 Abstract I

Abstract **I**

Présentation de l'entreprise et du Cadre **I**
0.3 L'entreprise ColibriTD I
0.4 Cadre de Stage III

Introduction **1**

1 Les réseaux de neurones **3**
1.1 Perceptron 3
1.2 Réseau de neurones 5
1.2.1 Architecture d'un réseau de neurones artificiels 6
1.2.2 Fonctionnement d'un réseau de neurones 6

2 Principes de l'informatique quantique et de l'algorithme variationnel quantique (VQA) **8**
2.1 Notion de qubit (quantum bit) 8
2.1.1 Les principes de l'informatique quantique 9
2.1.2 Porte quantique 10
2.1.3 Porte de Toffoli et Théorème d'universalité 13
2.1.4 Algorithme de Deutsch-Jozsa 14
2.2 Algorithme quantique variationnel (VQA) 17
2.2.1 Principe de fonctionnement 17

3 Physics-informed neural networks (PINNs) **20**
3.1 Principe de fonctionnement des PINNs 21
3.1.1 Fonction d'activation 21
3.1.2 Les conditions aux bords 22

3.1.3 Méthode d'optimisation 22
3.2 Cas pratique 23
3.2.1 Cas pratique 1 23

4 L'algorithme quantique H-DES (Hybrid Differential Equation Solver) 26
4.1 Fonctionnemen de HDES 26
4.1.1 Ansatz et Circuit variationnel quantique utilisés 27
4.1.2 Construction de la fonction solution 28
4.1.3 Défintion de la fonction coût 29
4.1.4 Application 31

5 Comparaison des deux méthodes (PINNs et H-DES) : Cas des équations de la flamme 33
5.1 Résolution de l'équation de la flamme avec les PINNs 33
5.1.1 Etude des erreurs sur l'ensemble de test 39
5.2 Résolution de l'équation de la Flamme avec H-DES 45
5.3 Comparaison des deux méthodes cas de l'équation de la flamme 52

Conclusion et Perspectives 55

Bibliographie 56

Présentation de l'entreprise et du Cadre de Stage

0.3 L'entreprise ColibriTD

ColibriTD est une entreprise créée en 2019 par le Dr Laurent Guiraud et Hacène GOUJIL. C'est une entreprise française dont le siège se trouve à Paris. ColibriTD développe des logiciels quantiques afin de permettre aux entreprises de tirer profit de l'informatique quantique en y facilitant l'accès. En effet le plus grand défi dans les domaines tels que la découverte de médicaments, la science des matériaux et la chimie est la limitation de la puissance de calcul. Les simulations de calcul à haute performance (HPC) peuvent prendre plusieurs jours, voire plusieurs semaines. L'informatique quantique a le potentiel de réduire ces temps de simulation de semaines ou de jours à seulement quelques minutes ou secondes, accélérant ainsi de manière significative le rythme de l'innovation. Plusieurs acteurs œuvrent sur le marché du quantique en proposant différents matériels (ordinateurs quantiques) et des langages spécifiques pour y accéder. Cette fragmentation du marché rend difficile l'accès au quantique.
Pour résoudre ce problème et rendre l'accès facile à l'outil quantique, l'entreprise ColibriTD propose deux produits : le **QUICK** et le **MPQP**.
Le QUICK (QUantum Innovative Computing Kit) est une plateforme qui intègre différents algorithmes quantiques pour des cas d'utilisation comme la déformation de matériel, la combustion... Pour rendre cette solution accessible sur différents matériels, ColibriTD a mis en place le MPQP (Multi Platform Quantum Programming). Le langage multiplateforme fait partie intégrante du QUICK.

FIGURE 1 – La solution QUICK

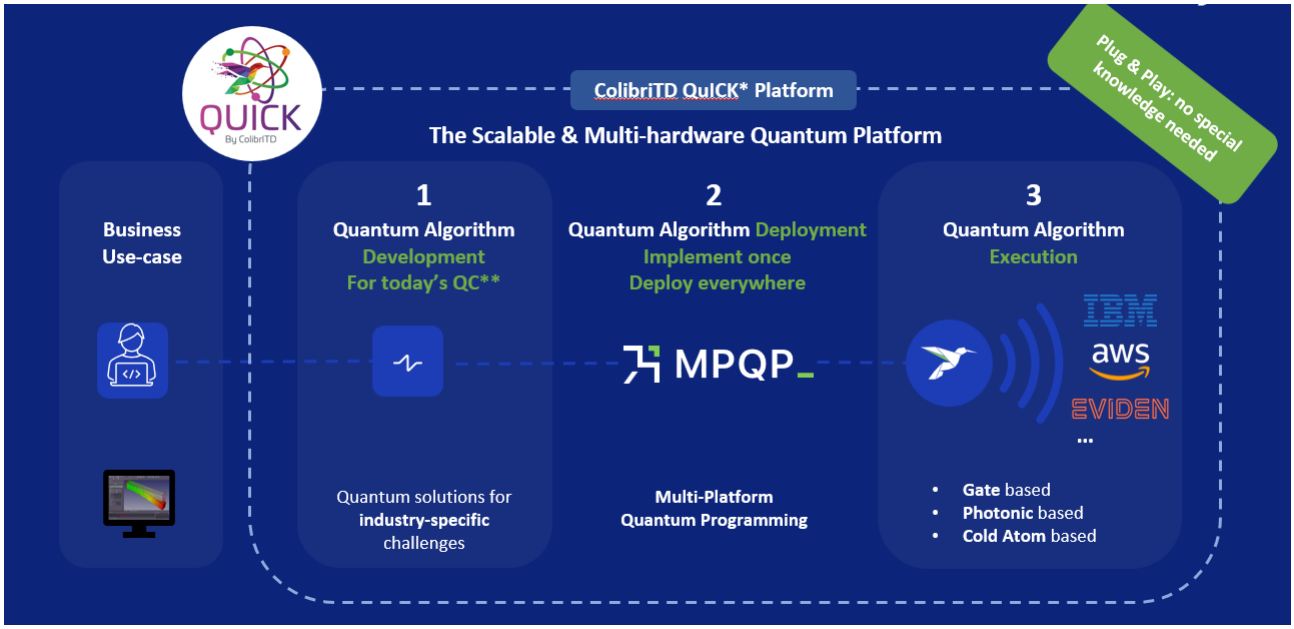


FIGURE 2 – Flux de travail du QUICK

L'utilisation de la plateforme QUICK ne requiert aucune connaissance particulière ce qui rend l'accès à l'informatique quantique facile.

ColibriTD se positionne sur le marché de la simulation industrielle avec des utilisateurs dans l'industrie de la défense et de l'aérospatiale. Elle veut tirer parti du plein potentiel de l'informatique quantique en l'intégrant aux outils de simulation déjà existants. ColibriTD prévoit aussi d'étendre son marché en s'attaquant à d'autres domaines nécessitant la simulation comme la santé, l'énergie, etc. Le marché des logiciels de simulation est un marché en pleine expansion avec une estimation de 6.26 milliards de dollars en 2020 et une prévision de 13.45 milliards pour 2026. ColibriTD compte tirer profit de cette expansion en rendant les simulations plus rapides, plus précises et moins coûteuses.

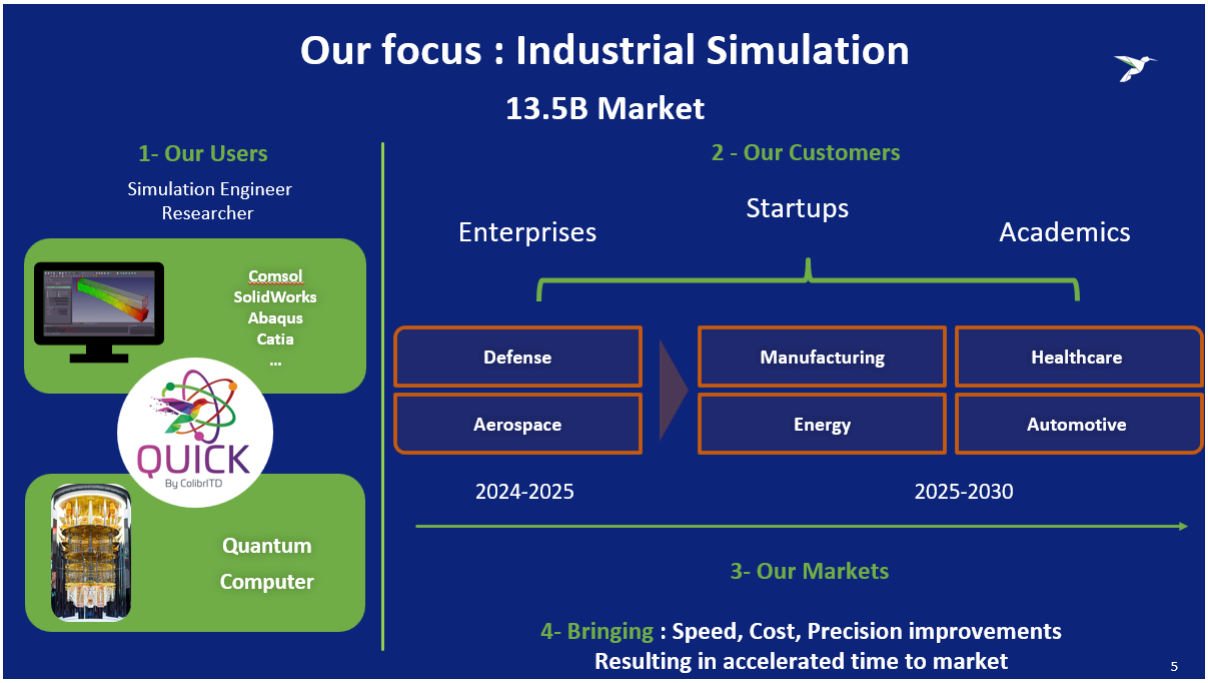


FIGURE 3 – le marché de ColibriTD

0.4 Cadre de Stage

L'entreprise ColibriTD a une équipe dynamique constituée de chercheurs, de consultants et de stagiaires. Elle a plus de 15 experts en infomatique quantique dont 80% sont détenteurs d'un PhD. Pour mener à bien le développement de ses produits, ColibriTD est subdivisé en départements. Nous avons un département qui développe le langage MPQP et un département qui met en oeuvre des solutions pour les cas d'utilisation du QUICK. La plupart des cas d'utilisation comme la déformation de matériel, la combustion... se traduisent par la résolution d'équation aux dérivées partielles. Ainsi le dernier département

travaille essentiellement à la mise en oeuvre d'algorithme quantique pour la résolution des équations aux dérivées partielles comme HDES (Hybrid Differential Equation Solver). Mon stage s'est déroulé auprès du département chargé de la résolution des équations aux dérivées partielles.

Introduction

Les équations différentielles permettent de modéliser et de décrire des phénomènes naturels. Ce qui fait de leur résolution une importance cruciale pour la compréhension de nombreux phénomènes physiques. Face à l'incapacité de résoudre certaines équations de manière analytique (pour trouver des solutions exactes), des méthodes numériques (elles permettent de trouver des solutions approximatives avec des ordinateurs.) ont vu le jour. Mais très tôt elles montrent aussi leurs limites dans certains cas. En effet, ces méthodes numériques peuvent se montrer très coûteuses en ressources computationnelles pour des équations à haute dimension ou pour des simulations sur de longues périodes. Elles peuvent être aussi sensibles aux conditions initiales... Bref, elles rencontrent de nombreux défis. Les domaines comme les réseaux de neurones ou l'informatique quantique (nouvelle façon de faire de l'informatique en se basant sur les principes de la physique quantique) présentent des avantages certains ; comme la capacité de représenter des fonctions à haute dimension pour les réseaux de neurones ou la capacité d'effectuer des calculs exponentiellement plus rapides que l'informatique classique pour l'informatique quantique (on remarque que cela n'est possible que pour certains types de problèmes.). Ainsi, de nouvelles méthodes pour la résolution des équations différentielles ont vu le jour comme les Physics-Informed Neural Networks (Introduite par Raissi et al. [18]) ou certaines méthodes quantiques. Bien que ces méthodes peuvent présenter des limites, elles sont prometteuses pour attaquer certains problèmes devant lesquels les méthodes numériques se bloquent.

Les ordinateurs quantiques bien que prometteurs se heurtent à des problèmes comme la décohérence quantique, les erreurs... Pour mitiger ces problèmes, les chercheurs ont mis au point diverses techniques comme les algorithmes variationnels quantiques (VQA) dont le principe de fonctionnement est similaire à celui d'un réseau de neurones.

Dans ce mémoire, notre but est de comparer les deux méthodes de résolution des équations différentielles que sont les Physics-Informed Neural Networks (PINNs) et le Hybrid Differiential Equation Solver (H-DES) : Une méthode basée sur le principe du VQA développé par l'entreprise ColibriTD. Pour ce faire, nous structurons ce mémoire comme suit :

— Dans le chapitre 1 nous introduisons les réseaux de neurones artificiels ; leur archi-

tecture et leur fonctionnement.

— Le chapitre 2 est consacré à la présentation de l'informatique quantique et des principes qui les sous-tendent. Nous parlons aussi du théorème d'universalité qui montre qu'il existe un ensemble de portes quantiques pour réaliser tout algorithme quantique. Enfin, nous introduirons l'algorithme variationnel quantique qui tire profit de l'informatique quantique et classique.

— Dans le chapitre 3, nous parlerons des Physics-Informed Neural Networks. Nous présenterons leur fonctionnement puis différentes techniques d'initialisation, de choix des fonctions d'activation, de la méthode d'optimisation et de prise en compte des conditions initiales et aux bords.

— Nous présenterons dans le chapitre 4 l'algorithme hybride (utilise l'informatique quantique et classique) H-DES utilisant un circuit quantique variationnel.

— Dans le chapitre 5 nous étudions l'équation de la Flamme laminaire unidimensionnel. Nous résolvons cette équation en utilisant les Physics-Informed Neural Networks et le H-DES puis nous comparons ces 2 méthodes sur différents points comme le choix de la méthode d'optimisation, de la fonction coût, le nombre de paramètres...

Chapitre 1

Les réseaux de neurones

Dans leur article [3], *A Logical Calculus of the Ideas Immanent in Nervous Activity* publié en 1943, Warren McCullock et Walter Pitts ont établie le premier concept simplifié d'une cellule du cerveau (neurone). Ils ont décrit un neurone comme une simple porte logique avec une sortie binaire (0 ou 1). L'idée est que plusieurs signaux arrivent au niveau des neurones (plus précisément les dendrites), puis y sont accumulés ; si tous ces signaux dépassent un seuil alors le neurone est activé et une sortie est générée.
Quelques années après Franck Rosenblatt introduit la notion de perceptron dans son article [21](The Perceptron, a Perceiving and Recognizing Automation). Il introduit la règle du perceptron et propose un algorithme pour apprendre les poids optimaux qui seront multipliés par les entrées du perceptron dans le but de prendre une décision selon que le neurone est activé ou pas. Il jette ainsi les bases de l'apprentissage automatique.
Dans ce chapitre, nous allons commencer par introduire la notion de perceptron, puis de réseau de neurones et le résultat fondamental qui les sous-tend, c'est-à-dire le théorème d'approximation universelle.

1.1 Perceptron

Le problème du perceptron peut être formuler comme un problème de classification binaire où nos classes sont 0 et 1 (0 pour la classe négative et 1 pour la classe positive).

Soit $\phi(y)$ une fonction d'activation qui prend une combinaison linéaire d'un vecteur x (valeurs d'entrée du réseau) et d'un vecteur poids w.

On a $y = a_1x_1 + a_2x_2 + \cdots + a_nx_n$, avec

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, w = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

Selon que la sortie $\phi(y)$ est supérieur ou inférieur à une certaine valeur seuil θ définie, on prédit la classe 1 ou 0.

$$\phi(y) = \begin{cases} 1 & \text{si } y \geq \theta \\ 0 & \text{sinon} \end{cases}$$

Dans le cas où $\theta = 0$, $\phi(y)$ est également appelé la fonction marche de Heaviside.

$$H(y) = \begin{cases} 1 & \text{si } y \geq 0 \\ 0 & \text{sinon} \end{cases}$$

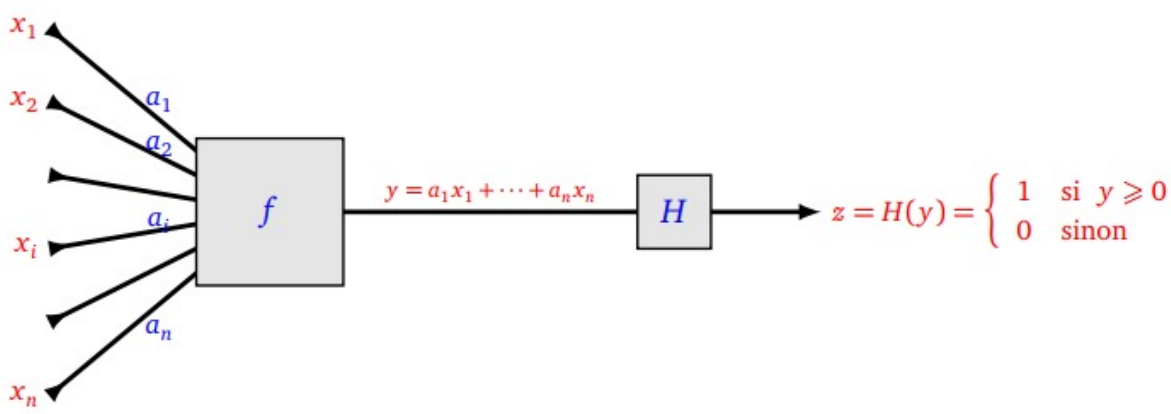


FIGURE 1.1 – Perceptron

Définition. *Un **perceptron** linéaire à n variables et de fonction d'activation la fonction marche de heaviside est la donnée de n coefficients réels $a_1, \ldots, a_n$ auxquels est associée la fonction $F : \mathbb{R} \to \mathbb{R}$ définie par $F = H \circ y$, c'est à dire :*

$$\begin{cases} F(x_1, \ldots, x_2) = 1 & \textit{si } a_1x_1 + \cdots + a_nx_n \geq 0, \\ F(x_1, \ldots, x_2) = 0 & \textit{sinon.} \end{cases}$$

De manière condensée un perceptron sera représenté sous la forme d'un neurone avec des poids sur les arêtes d'entrée.

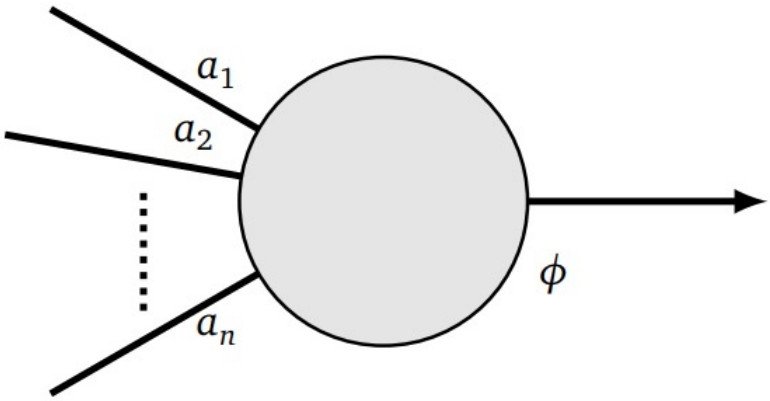


FIGURE 1.2 – Perceptron en forme de neurone

Remarque. *Un perceptron a des limites et ne peut pas représenter certaines fonctions complexes (exemple du OU exclusif). Alors comment mieux faire ? Pourquoi pas connecter plusieurs neurones ?*

1.2 Réseau de neurones

Définition. *Un **réseau de neurone** est la juxtaposition de plusieurs neurones structurés en couches et reliés entre eux.*

Nous avons différents types de réseau de neurones comme les réseaux de neurones artificiels (ANN), les réseaux de neurones convolutifs (CNN) pour le traitement d'images, ou les réseaux récurrents.
Nous nous focaliserons seulement sur les réseaux multicouches qui sont les types de réseaux souvent utilisés pour la résolution des équations différentielles.

Théorème 1. ***Théorème d'approximation universelle***
Toute fonction continue

$$f : [a, b] \to \mathbb{R}$$

peut être approchée d'aussi près que l'on veut par une fonction

$$F : [a, b] \to \mathbb{R}$$

réalisée par un réseau de neurones.

On ne démontrera pas ce théorème. Mais on peut noter que ce théorème reste vrai pour une fonction à plusieurs variables sous certains hypothèses.

Remarque. *En pratique il est plus difficile de trouver l'architecture adéquate pour approximer une fonction donnée car cela dépend de plusieurs facteurs comme la profondeur du réseau, la fonction d'activation, l'initialisation des poids, le nombre d'itérations, le choix d'optimiseur ... Nous verrons par la suite l'importance de ces facteurs dans notre problème.*

1.2.1 Architecture d'un réseau de neurones artificiels

Un réseau de neurone artificiel est constitué de différentes couches que sont :

— La couche d'entrée qui prend les données d'entrée ou d'entrainement.
— Les couches cachées qui sont les couches situées entre la couche d'entrée et de sortie. Le nombre de couches cachées dépend de la complexité du modèle qu'on veut construire et définit la profondeur du réseau.
— la couche de sortie constituée de un ou plusieurs neurones selon la sortie voulue.

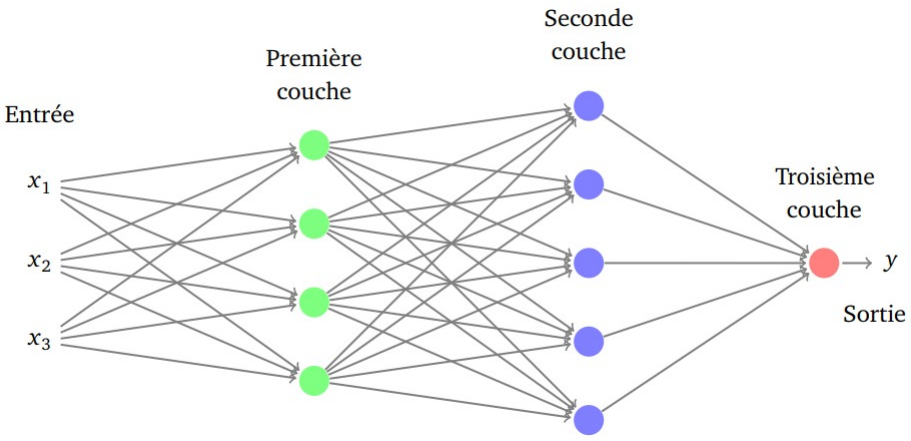


FIGURE 1.3 – Architecture d'un réseau de neurones artificiels

1.2.2 Fonctionnement d'un réseau de neurones

Soit un réseau de neurones artificiels d'ensemble d'entrainement $(x_i, y_i)_{i=1}^N$, de poids $w(a_1, a_2, \ldots, a_n)$, de fonction d'activation ϕ (nous supposerons la même fonction d'activation au niveau de chaque couche).
On peut décrire le fonctionnement de ce réseau à l'aide de cet algorithme simple.

1. Initialiser les poids et les biais aléatoirement ou en utilisant différentes stratégies d'initialisation.
2. Pour l'ensemble d'entrainement $(x_i, y_i)_{i=1}^N$ faire :

(a) Pour chaque couche l à n neurones

Propagation en avant :

i. Calculer la somme au niveau de chaque neurone de la couche l : $z_l = \sum_{j=1}^{n} a_{lj} \cdot x_{l-1,j} + b_l$

ii. Appliquer la fonction d'activation : $\phi(z_l)$

(b) Calculer l'erreur L entre la sortie prédite et la sortie réelle

Retropropagation :

3. Pour chaque couche :

(a) Calculer les gradients des poids et des biais.

(b) Mettre à jour les poids et les biais à l'aide de la descente de gradient : $a_{lj} = a_{lj} - \eta \cdot \frac{\partial L}{\partial a_{lj}}$; $b_l = b_l - \eta \cdot \frac{\partial L}{\partial b_l}$

Remarque. *L'algorithme de descente de gradient utilisé lors de la retropropagation pour mettre à jour les poids a différentes variantes selon l'optimiseur choisi. Ainsi on a des optimiseurs basés sur des gradients stochastiques comme Adam ou détermistes comme BFGS. Nous verrons dans le suite l'importance du choix dans notre problème.*

Chapitre 2

Principes de l'informatique quantique et de l'algorithme variationnel quantique (VQA)

Contrairement à l'informatique classique dans laquelle l'unité basique de l'information est le bit (0 ou 1), l'informatique quantique utilise plutôt le qubit (quantum bit) qui peut être dans l'état 0, 1 ou les deux à la fois (superposition d'états). C'est un domaine prometteur avec des applications potentielles dans la cryptographie, la simulation ou encore l'optimisation. Dans ce chapitre, nous introduisons la notion de qubit, des principes de l'informatique quantique puis des portes quantiques qui servent dans la construction des circuits quantiques. Pour finir, nous parlons de l'algorithme variationnel quantique. Ce chapitre est basé sur [1] et le cours [9]

2.1 Notion de qubit (quantum bit)

Définition 2. *Soit* $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ *et* $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ *une base orthonormale de* $\mathbb{C}^2$*, a et b des nombres complexes. Un qubit* $|\psi\rangle$ *est un vecteur normalisé de* $\mathbb{C}^2$ *définit par :* $|\psi\rangle = a|0\rangle + b|1\rangle$ *avec* $|a|^2 + |b|^2 = 1$

Remarque.

— $|.\rangle$ *se lit ket.*

— *si* $w = \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix} \in \mathbb{C}^n$ *on le notera* $|w\rangle$

— *le vecteur* $W^\dagger = (\overline{w_1}, \overline{w_2}, \dots, \overline{w_n})$ *se notera* $\langle w|$ *(*$\langle .|$ *se lit bra).*

Définition 3. *Soit $|\psi\rangle = a|0\rangle + b|1\rangle$ un qubit dans la base $(|0\rangle, |1\rangle)$. Mesurer $|\psi\rangle$ dans la base $(|0\rangle, |1\rangle)$ consiste à le projeter dans cette base. Le résultat sera 0 ou 1 et on aura une probabilité $|a|^2$ et $|b|^2$ d'avoir respectivement $|0\rangle$ ou $|1\rangle$*

2.1.1 Les principes de l'informatique quantique

Les principes de l'informatique quantique se basent sur les lois de la physique quantique.

Axiome 1. ***(Principe de Superposition)*** *Soit $|\psi\rangle$ un vecteur d'un espace vectoriel complexe muni du produit scalaire (on parle d'espace d'Hilbert) et $|e_1\rangle, |e_2\rangle, \ldots, |e_n\rangle$ une base orthonormale de cet espace. $|\psi\rangle$ peut être exprimé comme combinaison linéaire des états $|e_i\rangle$:*

$$|\psi\rangle = a_1|e_1\rangle + \cdots + a_n|e_n\rangle. \tag{2.1}$$

En particulier $\langle e_i, e_i\rangle = 1$ et $\langle e_i, e_j\rangle = 0$ si $i \neq j$

Axiome 2. *les a_i satisfont*

$$|a_1|^2 + \cdots + |a_2|^2 = 1 \tag{2.2}$$

$|a_i|^2$ représentent la probabilité d'observer l'état $|e_1\rangle$ après la mesure de $|\psi\rangle$ dans la base $|e_1\rangle, |e_2\rangle, \ldots, |e_n\rangle$.

Axiome 3. *Après la mesure, l'état $|\psi\rangle$ devient l'un des états $|e_i\rangle$ de la base $(|e_1\rangle, |e_2\rangle, \ldots, |e_n\rangle)$ avec la probabilité $|a_i|^2$.*

Définition 4. *Une transformation linéaire $f : \mathcal{H} \to \mathcal{H}$ est dite unitaire si et seulement si*

$$\langle f(v), f(w)\rangle = \langle v, w\rangle, \forall v, w \in \mathcal{H}$$

Si U_f est une matrice encodant la transformation linéaire f dans une base, alors

$${}^tU_f U_f = Id$$

Axiome 4. *Un état quantique évolue dans le temps selon une transformation unitaire c'est à dire*

$$|\psi(t)\rangle = U(t)|\psi(0)\rangle \tag{2.3}$$

2.1.2 Porte quantique

Un algorithme est la donnée d'une entrée à laquelle on applique une succession d'opérations pour avoir la sortie voulue. En informatique quantique, ces données sont encodées sous forme d'état quantique (c'est-à-dire un vecteur complexe normalisé), les successions d'opérations correspondent à l'application d'une matrice unitaire à l'état de départ et la sortie à des états quantiques qui pourront être mesurés pour trouver l'information classique. Nous voyons ci-dessous quelques exemples de portes quantiques et les opérations qui leur sont associées.

Porte d'Hadamard H

Elle est utilisée pour créer des superpositions d'états.
La porte Hadamard (H) est définie par la matrice suivante :

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Application sur l'état $|0\rangle$

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

En appliquant la porte Hadamard, nous avons :

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Ce qui donne l'état :

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

Application sur l'état $|1\rangle$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

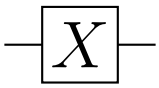
En appliquant la porte Hadamard, nous avons :

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Ce qui donne l'état :

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Porte de Pauli-X ou NOT gate



La porte X (ou Pauli-X) est définie par la matrice suivante :

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

L'état $|0\rangle$ est représenté par le vecteur colonne :

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

En appliquant la porte X, nous avons :

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Ce qui donne l'état :

$$X|0\rangle = |1\rangle$$

Application sur l'état $|1\rangle$
L'état $|1\rangle$ est représenté par le vecteur colonne :

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

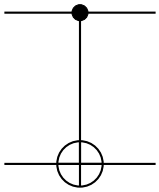
En appliquant la porte X, nous avons :

$$X|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Ce qui donne l'état :

$$X|1\rangle = |0\rangle$$

Porte CNOT (Controlled-NOT gate)



La porte CNOT est une porte pour 2-qubits et s'appelle *controlled-not gate* La matrice de la porte CNOT est :

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Considérons les états quantiques suivants :

1. État Initial $|00\rangle$:

$$\text{CNOT}|00\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle$$

2. État Initial $|01\rangle$:

$$\text{CNOT}|01\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = |01\rangle$$

3. État Initial $|10\rangle$:

$$\text{CNOT}|10\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |11\rangle$$

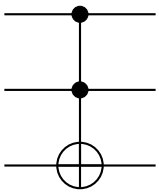
4. État Initial $|11\rangle$:

$$\text{CNOT}|11\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |10\rangle$$

2.1.3 Porte de Toffoli et Théorème d'universalité

Porte de Toffoli

La porte Toffoli, également appelée porte CNOT-CNOT, est une porte quantique à trois qubits où deux qubits agissent comme qubits de contrôle et le troisième comme qubit cible.



La matrice unitaire associée à la porte Toffoli est :

$$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Voici un tableau montrant l'effet de la porte Toffoli sur les états de base de trois qubits. La porte Toffoli applique une opération X (NOT) au troisième qubit uniquement lorsque les deux premiers qubits (qubits de contrôle) sont dans l'état $|1\rangle$.

État Initial	**État Après Toffoli**
$\|000\rangle$	$\|000\rangle$
$\|001\rangle$	$\|001\rangle$
$\|010\rangle$	$\|010\rangle$
$\|011\rangle$	$\|011\rangle$
$\|100\rangle$	$\|100\rangle$
$\|101\rangle$	$\|101\rangle$
$\|110\rangle$	$\|111\rangle$
$\|111\rangle$	$\|110\rangle$

On pourrait se poser la question ; si pour toute fonction exécutable par un circuit classique (ordinateur classique) on peut associer ou trouver un circuit quantique pour

l'exécuter. En remarquant que la porte de Toffoli est capable de simuler les portes classiques **AND**, **OR** et **NOT**, le **théorème d'universalité** ci-dessous montre que cela est possible en choisissant des portes de Toffoli.

Théorème 5 (Théorème d'universalité)**.** *La porte de Toffoli est universelle : n'importe quelle fonction logique* $f : \{0,1\}^n \to \{0,1\}$ *peut être réalisée par un circuit quantique ne comportant que des portes de Toffoli.*

Remarque. *Une conséquence du théorème d'universalité est qu'à une fonction* f*, on peut associer un circuit appelé oracle de* f *noté* O_f*. L'oracle de* f *est un circuit quantique dont ont défini uniquement l'entrée et la sortie. Elle fonctionne comme une boite noire. On a besoin de connaître ce qui se passe à l'intérieur. Ci-dessous le fonctionnement d'un oracle. On peut voir qu'il transforme l'entrée* y *en* $y \otimes f(x)$ *et garde intacte* x*.*

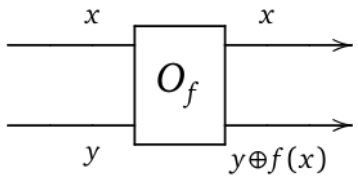


FIGURE 2.1 – Définition de l'oracle d'une fonction f

2.1.4 Algorithme de Deutsch-Jozsa

Nous introduisons dans cette partie un des algorithmes quantiques les plus populaires. Le problème est énoncé comme suit :
Soit $f : \{0,1\}^n \to \{0,1\}$ une fonction booléenne qui est soit constante ou équilibrée (c'est à dire s'il y a autant de n-uplets $(x_1, \ldots, x_n)$ tels que $f(x_1, \ldots, x_n) = 0$ que de n-uplets $(x_1, \ldots, x_n)$ que $f(x_1, \ldots, x_n) = 1$). Le problème est de trouver si f est constante ou équilibrée.

Solution classique

Pour résoudre ce problème la complexité du meilleur algorithme d'un ordinateur classique est d'ordre $O(2^n)$. Ci-dessous un algorithme qui résout le problème avec une complexité de $2^{n-1} + 1$.

Algorithme

— On évalue f sur $2^{n-1} + 1$ termes
— Si toutes ces valeurs sont égales alors f est constante, sinon elle est équilibrée.

Solution quantique

Voici le circuit de l'algorithme quantique qui résout le problème.

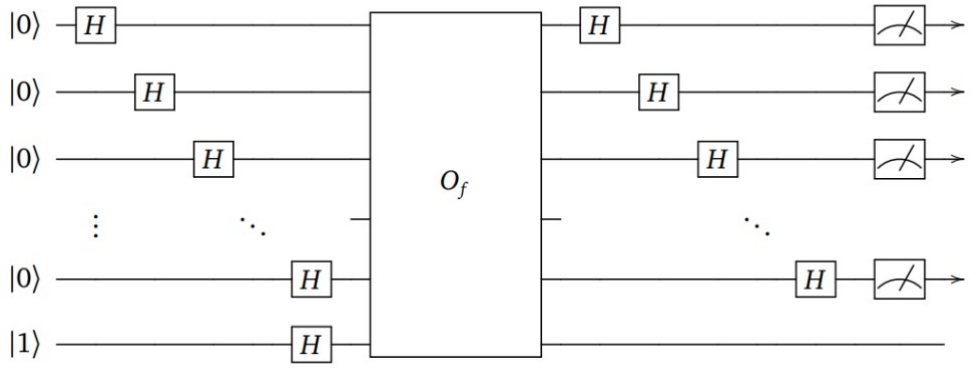


FIGURE 2.2 – circuit de l'algorithme Deutsch-Jozsa

On initialise les n premières lignes en $|0\rangle$ et la dernière en $|1\rangle$ puis on applique la transformation de Hadamard. Après on a l'oracle associé à f. Enfin, on applique de nouveau une transformation de Hadamard sur les n premières lignes, puis on mesure uniquement les n premières lignes. La complexité est $O(1)$.

Démonstration de l'algorithme

Soit $|\underline{0}\rangle = |0\dots0\rangle$

Après la transformation de Hadamard sur $|\underline{0}\rangle$ on a :

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}}\left(|0\dots00\rangle + |0\dots01\rangle + \cdots + |1\dots11\rangle\right) \tag{2.4}$$

C'est à dire en écriture décimale on a :

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}}\sum_{\ell=0}^{2^n-1}|\ell\rangle \tag{2.5}$$

— Initialisation.

$$|\psi_0\rangle = |0\dots0\rangle \otimes |1\rangle = |\underline{0}\rangle \otimes |1\rangle.$$

On initialise les n premières lignes ($|0\dots0\rangle$) et la dernière ligne en ($|1\rangle$).

— Transformation de Hadamard.

$$|\psi_1\rangle = H^{\otimes(n+1)}|\psi_0\rangle \tag{2.6}$$
$$= H^{\otimes n}|\underline{0}\rangle \otimes H|1\rangle \tag{2.7}$$
$$= \frac{1}{\sqrt{2^n}} \sum_{\ell=0}^{2^n-1} |\underline{\ell}\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \tag{2.8}$$

— Oracle.

$$\begin{aligned} |\psi_2\rangle &= O_f|\psi_1\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{\ell=0}^{2^n-1} |\underline{\ell}\rangle \cdot \frac{1}{\sqrt{2}} (|0 \oplus f(\underline{\ell})\rangle - |1 \oplus f(\underline{\ell})\rangle) \\ &= \frac{1}{\sqrt{2^n}} \sum_{\ell=0}^{2^n-1} (-1)^{f(\underline{\ell})} |\underline{\ell}\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \end{aligned}$$

On a utilisé :

$$|0 \oplus f(\underline{\ell})\rangle - |1 \oplus f(\underline{\ell})\rangle = \begin{cases} |0\rangle - |1\rangle & \text{si } f(\underline{\ell}) = 0 \\ -(|0\rangle - |1\rangle) & \text{si } f(\underline{\ell}) = 1 \end{cases} = (-1)^{f(\ell)} (|0\rangle - |1\rangle) .$$

— Transformation de Hadamard.

$$\begin{aligned} |\psi_3\rangle &= H^{\otimes n} \left(\frac{1}{\sqrt{2^n}} \sum_{\ell=0}^{2^n-1} (-1)^{f(\underline{\ell})} |\underline{\ell}\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\ &= \frac{1}{\sqrt{2^n}} \sum_{\ell=0}^{2^n-1} (-1)^{f(\underline{\ell})} H^{\otimes n} (|\underline{\ell}\rangle) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\ &= \frac{1}{\sqrt{2^n}} \sum_{\ell=0}^{2^n-1} (-1)^{f(\underline{\ell})} \left(\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} (-1)^{\underline{\ell} \cdot k} |k\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\ &= \frac{1}{2^n} \sum_{k=0}^{2^n-1} |k\rangle \sum_{\ell=0}^{2^n-1} (-1)^{f(\underline{\ell}) + \underline{\ell} \cdot k} \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \end{aligned}$$

Trouver la probabilité de mesurer $0.0 \dots 0$ pour les n premiers qubits, c'est trouver le coefficient $\alpha \in \mathbb{C}$ devant le qubit $|0.0 \dots 0\rangle \cdot \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Autrement dit, il s'agit de trouver le coefficient correspondant à $k = 0$:

$$\alpha = \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} (-1)^{f(\underline{\ell}) + \ell \odot 0}.$$

Comme $\underline{\ell} \odot 0 = 0$ alors :

$$\alpha = \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} (-1)^{f(\underline{\ell})}.$$

— Si f est constante, par exemple $f(\underline{\ell}) = 0$, pour tout ℓ, alors :

$$\alpha = \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} (-1)^0 = \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} 1 = 1.$$

Comme le qubit $|\psi_3\rangle$ est normalisé, et que $\alpha = 1$ alors les autres coefficients des termes de $|\psi_3\rangle$ sont tous nuls. Ainsi dans ce cas $|\psi_3\rangle = |0.0\ldots0\rangle \cdot \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ et la probabilité de mesurer $0.0\ldots0$ sur les n premiers qubits est 1.

De même si f était constante égale à 1, alors on trouverait $\alpha = -1$ et $|\psi_3\rangle = -|0.0\ldots0\rangle \cdot \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ et la probabilité de mesurer $0.0\ldots0$ sur les n premiers qubits est également 1.

— Si f est équilibrée, il y a autant de ℓ avec $f(\underline{\ell}) = 0$ que de $\underline{\ell}$ avec $f(\underline{\ell}) = 1$, ainsi il y a autant de ℓ avec $(-1)^{f(\underline{\ell})} = +1$ que $(-1)^{f(\underline{\ell})} = -1$. (On rappelle $(-1)^p = \pm 1$.) Ainsi la somme des $(-1)^{f(\underline{\ell})}$ est nulle et donc $\alpha = 0$. La probabilité de mesurer $0.0\ldots0$ sur les n premiers qubits est donc 0.

Conclusion : si la mesure sur les n premiers qubits est $0.0\ldots0$ alors la fonction f est constante, sinon c'est qu'elle est équilibrée.

2.2 Algorithme quantique variationnel (VQA)

Les ordinateurs quantiques actuels ont des limitations au niveau du nombre de qubits, de la profondeur du circuit à cause des problèmes de bruit. Les algorithmes quantiques variationnels qui combinent les ordinateurs quantiques et classiques ont immergés pour essayer de répondre à ces problèmes [25] [16].

2.2.1 Principe de fonctionnement

On peut décrire le fonctionnement du VQA comme suit [11] :

— **Initialisation :** On commence par initialiser un état par défaut $|0\rangle$ qu'on transforme en un état non paramétré (état de référence) $|\rho\rangle$ par application d'une matrice unitaire U_R. On a $U_R|0\rangle = |\rho\rangle$.

— **Préparation de l'ansatz :** On définit une forme variationnelle $U_V(\vec{\theta})$ qui représente une collection d'états paramétrés. Elle nous permettra d'atteindre l'état voulu $|\psi(\theta)\rangle$. On définit aussi $U_A(\vec{\theta}) = U_V(\vec{\theta})U_R$ une combinaison de l'état de référence

et la forme variationnelle.

$$\begin{aligned} |0\rangle \xrightarrow{U_R} U_R|0\rangle = |\rho\rangle &\xrightarrow{U_V(\vec{\theta})} U_A(\vec{\theta})|0\rangle \\ &= U_V(\vec{\theta})U_R|0\rangle \\ &= U_V(\vec{\theta})|\rho\rangle \\ &= |\psi(\vec{\theta})\rangle \end{aligned}$$

Un système de n qubits possède un nombre immense d'états quantiques distincts. Il faudrait un nombre énorme de paramètres pour l'explorer complètement. Quantitativement, sa dimensionnalité est $D = 2^n$. Pour aggraver les choses, la complexité en temps d'exécution des algorithmes de recherche, et d'autres similaires, croît de manière exponentielle avec cette dimensionnalité, un phénomène souvent appelé la "malédiction de la dimensionnalité" (terme inventé par Richard Bellman en 1961). Pour ajouter, un circuit profond peut avoir des problèmes de bruits (erreurs) qui affectent la précision des mesures. Tout ceci amène à faire un compromis pour le choix de notre ansatz.

Nous avons des *ansatz heuristique* (fait un compromis entre vitesse d'exécution, exactitude du résultat et la profondeur du circuit) ; des *hardware-efficient ansatz* (prend en compte le matériel quantique et la profondeur du circuit pour éviter les erreurs) ou même *problem-specific*(c'est à dire qui utilise une connaissance de notre problème pour restreindre l'espace de recherche).

— **Evaluer la fonction coût :** Nous encoderons notre problème en une fonction coût $C(\vec{\theta})$ qui sera évalué par l'ordinateur quantique.

En physique classique pour décrire l'état d'une voiture en mouvement on utilise des propriétés comme la vitesse, la positon... En mécanique quantique on utilise la notion d'observable pour décrire l'état d'un système quantique. Un observable est un opérateur linéaire hermetien qui agit sur un état. Dans le cas où l'observable est l'énergie (Hamiltonien $\hat{\mathcal{H}}$), trouver l'énergie propre reviendrait à trouver :

$$\min_{\vec{\theta}} \langle\psi(\vec{\theta})|\hat{\mathcal{H}}|\psi(\vec{\theta})\rangle$$

Evaluer le coût c'est évaluer $C(\vec{\theta}) = \langle\psi(\vec{\theta})|\hat{\mathcal{H}}|\psi(\vec{\theta})\rangle$.

— **Optimisation des paramètres :** Après évaluation, la fonction coût est donnée à un ordinateur classique qui se chargera d'optimiser le coût et trouver les nouveaux paramètres de l'ansatz.

— **Mettre à jour l'ansatz et continuer :** On recommence le processus jusqu'à ce

qu'on trouve les paramètres $\vec{\theta}^*$ optimaux.

$$|\psi(\vec{\theta})\rangle = U_A(\vec{\theta})|0\rangle.$$

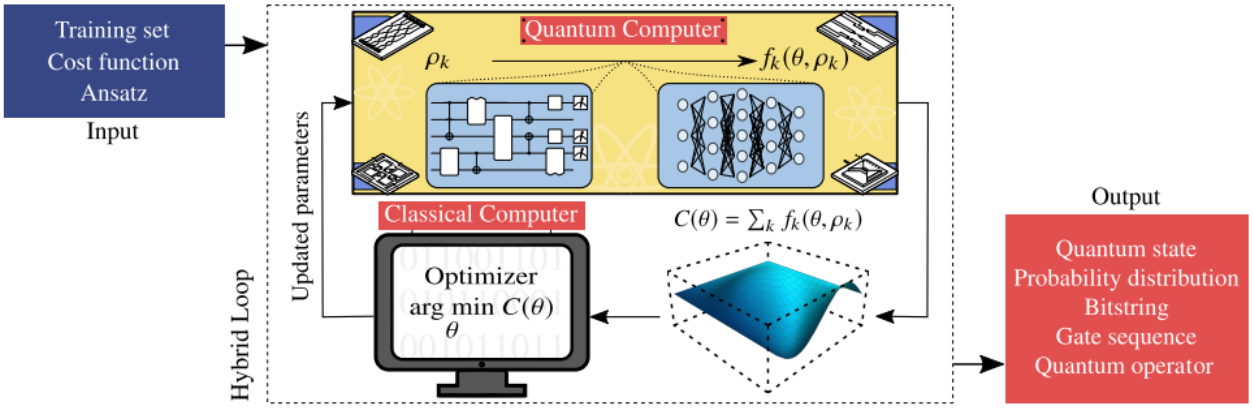


FIGURE 2.3 – diagramme du fonctionnement d'un VQA [16]

Sur cette figure, les entrées d'un algorithme quantique variationnel (VQA) sont : une fonction de coût $C(\theta)$, où θ est un ensemble de paramètres qui encode la solution au problème, un ansatz dont les paramètres sont entraînés pour minimiser le coût, et (éventuellement) un ensemble de données d'entraînement $\{\rho_k\}$ utilisées pendant l'optimisation.

De plus, l'ansatz est représenté par un circuit quantique paramétré (à gauche), qui est analogue à un réseau de neurones (également représenté schématiquement à droite). À chaque itération de la boucle, on utilise un ordinateur quantique pour estimer efficacement le coût (ou ses gradients). Ces informations sont ensuite fournies à un ordinateur classique qui utilise des optimiseurs pour naviguer dans l'espace du coût $C(\theta)$ et résoudre le problème d'optimisation. Une fois une condition de terminaison atteinte, le VQA fournit une estimation de la solution au problème. La forme de la sortie dépend de la tâche précise à accomplir. La boîte rouge indique certains des types de sorties les plus courants (état quantique, distribution probabilité...).

Chapitre 3

Physics-informed neural networks (PINNs)

L'apprentissage profond a eu des succès dans différents domaines comme la vision par ordinateur, le traitement de langage, la théorie des jeux, les tâches de régression... Il commence à être utilisé aussi pour des problèmes de mathématiques appliqués comme les équations aux dérivées partielles (EDP). Tout cela est rendu possible grâce au théorème d'approximation universelle.
Les Physics-Informed Neural Networks (PINNs) [18] sont des techniques d'apprentissage automatique non supervisées permettant de résoudre des problèmes impliquant les équations aux dérivées partielles. Les PINNs arrivent à approximer les solutions d'une EDP en entrainant un réseau de neurones à minimiser une certaine fonction coût. Ils permettent d'intégrer directement des lois physiques (souvent représentées sous forme d'EDP) dans la structure du réseau neuronal. En utilisant les lois physiques comme contrainte, l'expérience montre que les PINNs nécessitent moins de données d'entraînement ; ce qui est intéressant dans les contextes où les données sont rares. Les PINNs constituent aussi une alternative pour la résolution de certaines EDP non-linéaires complexes qui sont difficiles à résoudre pour les méthodes numériques classiques. Ils sont flexibles et peuvent être utilisés dans divers domaines comme la mécanique des fluides, l'électromagnétisme, la dynamique des structures... Ils sont aussi très efficaces dans les problèmes inverses où l'on cherche à déterminer des paramètres du système en fonction des données. Dans ce chapitre, nous donnons un aperçu du fonctionnement de PINNs avec quelques applications sur des EDP.

3.1 Principe de fonctionnement des PINNs

Soient les équations différentielles de la forme :

$$\mathcal{F}[u(z);\gamma] = f(z), \quad z \in \Omega, \tag{3.1}$$

$$\mathcal{B}[u(z)] = g(z), \quad z \in \partial\Omega$$

définie sur $\Omega \subset \mathbf{R}$ avec pour bord $\partial\Omega$ et $z = [x_1, \ldots, x_{d-t}, t]$ le vecteur espace-temps. u représente la solution inconnue, γ les paramètres du système physique, f est une fonction identifiant les données du problème et $\mathcal{F}$ est un opérateur différentiel. $\mathcal{B}$ est un opérateur indicant les conditions initiales ou les conditions aux bords.
Le but de ce problème peut être de déterminer la fonction u pour tous les z pour un γ spécifique ou déterminer γ avec les données (on parlera de problème inverse).
La méthode des PINNs consiste à prédire u avec un réseau de neurones en l'approximant avec une fonction $\hat{u}_\theta(z)$ où θ est l'ensemble des paramètres du réseau.
Le réseau de neurones approxime la solution en trouvant θ en minimisant une fonction coût qui dépend de l'équation différentielle $\mathcal{L}_\mathcal{F}$, les conditions aux bords $\mathcal{L}_\mathcal{B}$, et quelques données connues $\mathcal{L}_{data}$ [23] [24].

$$\theta^* = \arg\min_\theta \left(\omega_F \mathcal{L}_F(\theta) + \omega_B \mathcal{L}_B(\theta) + \omega_d \mathcal{L}_{\text{data}}(\theta)\right). \tag{3.2}$$

Le calcul des gradients par rapport aux valeurs d'entrée z ou les paramètres θ du réseau de neurones se fait à l'aide de la différentiation automatique [7].
Les PINNs ont été aussi implémentés avec diverses techniques de deep learning comme CNN(Réseau de neurone Convolutionel) [5], RNN(Réseau de neuronne Récurrent) [26] et GAN(réseau de Génératif) [28] mais la technique la plus implémentée est celle utilisant les réseaux à propagation en avant (perceptrons multicouches).
Nous étudierons dans ce rapport les PINNs avec les perceptrons multicouches.

3.1.1 Fonction d'activation

Dans la majorité des cas, la fonction d'activation utilisée dans les réseaux de neurones y compris pour les PINNs sont les fonctions comme **Sigmoid**, **RELU** et **Tanh**. Mais la question de leur adaptation se pose pour certains types de problèmes. Nous verrons par exemple que pour notre problème de la flamme, ces fonctions ne sont pas particulièrement adaptées. Pour résoudre ces problèmes, de nouvelles recherches préconisent d'entraîner sa propre fonction d'activation comme *Swish* définit comme $x.Sigmoid(\beta x)$ [19].

3.1.2 Les conditions aux bords

Les conditions peuvent être encodées de manière stricte en construisant une fonction à partir du modèle produit à chaque itération, ou peuvent être introduite avec une loss de manière plus souple. Avec cette dernière méthode, en plus, du fait que les conditions ne soient pas toujours bien atteintes, on a un problème dans l'efficacité de l'entraînement. Puisque avec plusieurs fonctions coût, certaines convergent plus rapidement d'autres lors de l'entraînement. Pour mitiger ce problème, on essaie de mettre des poids pour chaque fonction coût, mais il n'y a pas une solution spécifique pour savoir réellement comment ajuster ces poids. Les conditions strictes consistent à construire à partir du modèle produit par le réseau une solution sous la forme

$$\tilde{u} = \mathcal{N}(z, \hat{u}(z))$$

où $\hat{u}(z)$ est la fonction modéliser par le réseau de neurones (au ime-itérations) [2]. $\mathcal{N}(.)$ doit être construit de sorte à vérifier correctement les conditions.

Cette dernière méthode est surtout utilisée pour avoir une stricte vérification des conditions aux bords. La construction d'une nouvelle fonction pour une vérification stricte aussi pose problème des fois. Avec cette méthode, les modèles peuvent ne pas bien prédire les valeurs sur le reste du domaine.

3.1.3 Méthode d'optimisation

Les optimiseurs les plus utilisés pour les PINNs sont l'optimiseur Adam, et L-BFGS (limited-memory Broyden–Fletcher–Goldfarb–Shanno). Le choix de l'optimiseur peut agir sur la capacité à trouver un optimum.

La figure ci-dessous montre l'architecture des PINNs et leur fonctionnement.

On voit sur cette figure 2 composantes : le réseau de neurone et la partie seconde partie spécifique aux PINNs composée des dérivées partielles de u pour évaluer les fonctions coûts. Après nous avons un mécanisme qui minimise le coût avec un optimiseur et met à jour θ.

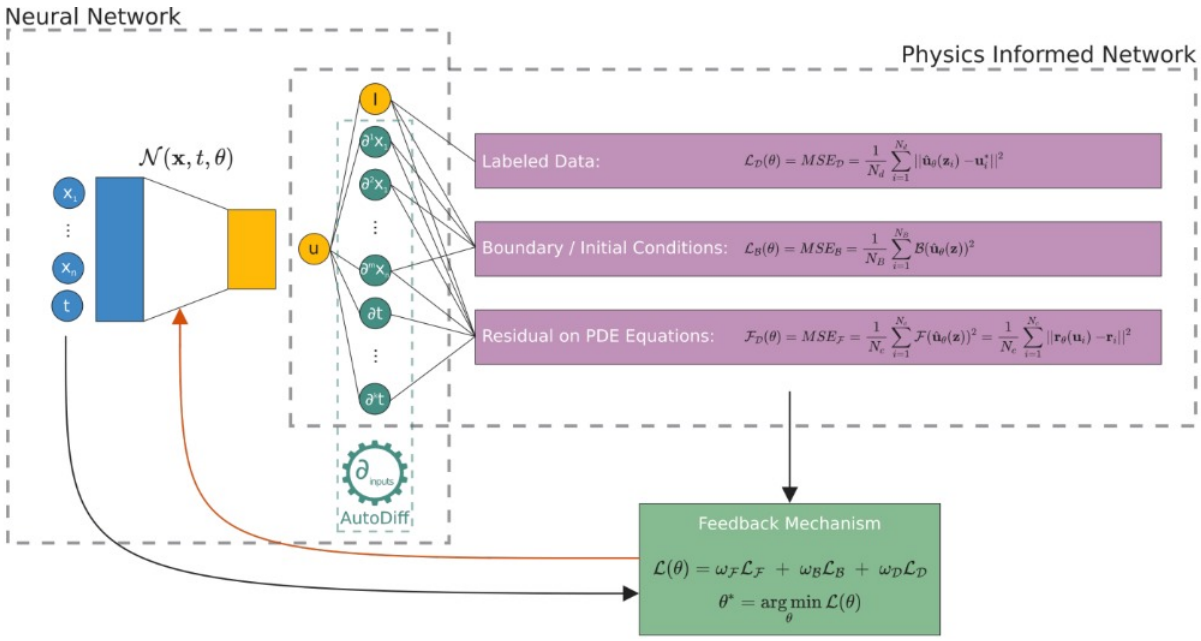


FIGURE 3.1 – Fonctionnement des PINNs [23]

.

3.2 Cas pratique

3.2.1 Cas pratique 1

Pour notre cas pratique, nous considérons le sytème suivant :

$$\begin{cases} \frac{df(x)}{dx} - 5 = 0 \\ \frac{dg(x)}{dx} - f(x) - 5 = 0 \end{cases} \tag{3.3}$$

$$\text{Conditions initiales : } \begin{cases} f(0) = 0 \\ g(0) = 0 \end{cases} \tag{3.4}$$

Les solutions sont :

$$f = 5x \tag{3.5}$$

$$g = 2.5x^2 + 5x \tag{3.6}$$

avec $f(0)$ et $g(0)$, définis sur l'intervalle $[0, 1]$. Pour résoudre ce problème nous allons choisir 20 points pour notre entraînement réparti uniformément sur l'intervalle $[0, 1]$ (équidistants).

Data: Choisir 20 points dans l'intervalle [0, 1] pour l'entrainement et 100 points entre [0, 1] pour les données de test

Result: La sortie sera une approximation des fonctions $f(x)$ et $g(x)$ pour les points test (ici le couple$(f(x) = 5x,\ g(x) = 2.5x^2 + 5x)$)

1 Choix d'un réseau de neurones entièrement connecté avec 1 entrée, 2 couches cachées avec 4 neurones chacun et 2 sorties (chaque sortie correspondant à une fonction solution)

2 Choisir tanh comme fonction d'activation et une initialisation utilisant Glorot uniform (technique d'initialisation inventée par xavier glorot pour lutter contre les problèmes de disparition et explosion du gradient).

3 Formuler le coût à minimiser comme une somme de

$$\mathcal{L} = \mathcal{L}_{r1}(\theta) + \mathcal{L}_{r2}(\theta) + \mathcal{L}_{bc}(\theta) \tag{3.7}$$

avec

$$\mathcal{L}_{r1} = \frac{1}{N}\sum_{i=1}^{N} |\frac{d\hat{f}}{dx}(x_i) - 5|^2$$

$$\mathcal{L}_{r1} = \frac{1}{N}\sum_{i=1}^{N} |\frac{d\hat{g}}{dx}(x_i) - \hat{f}(x_i) - 5|^2$$

$$\mathcal{L}_{bc}(\theta) = \frac{1}{2}(|\hat{f}(0) - 1|^2 + |\hat{g}(0) - 1|^2)$$

Minimiser la fonction coût en utilisant l'optimizer Adam

Après différents essais, nous sommes arrivés à bien prédire les 2 solutions avec seulement 11 neurones et 6 000 itérations. Pour le test, nous prenons 100 points. Nous avons pour notre meilleur modèle le coût de l'entraînement à $5.64 * 10^{-3}$ et le coût sur les données de test $4.42 * 10^{-3}$. Nous pouvons diminuer les coûts en augmentant soit le nombre de neurones ou le nombre d'itérations, mais cela ne nous semble pas nécessaire puisque comme nous le verrons avec le résultat (figure 3.3) nous arrivons déjà à approcher très bien nos 2 fonctions $f(x)$ et $g(x)$. Nous utilisons l'optimiseur adam [12] et nous utilisons la technique d'initialisation de Xavier Glorot [14], [27], [6](permet de lutter contre le problème des gradients qui disparaissent ou explosent).

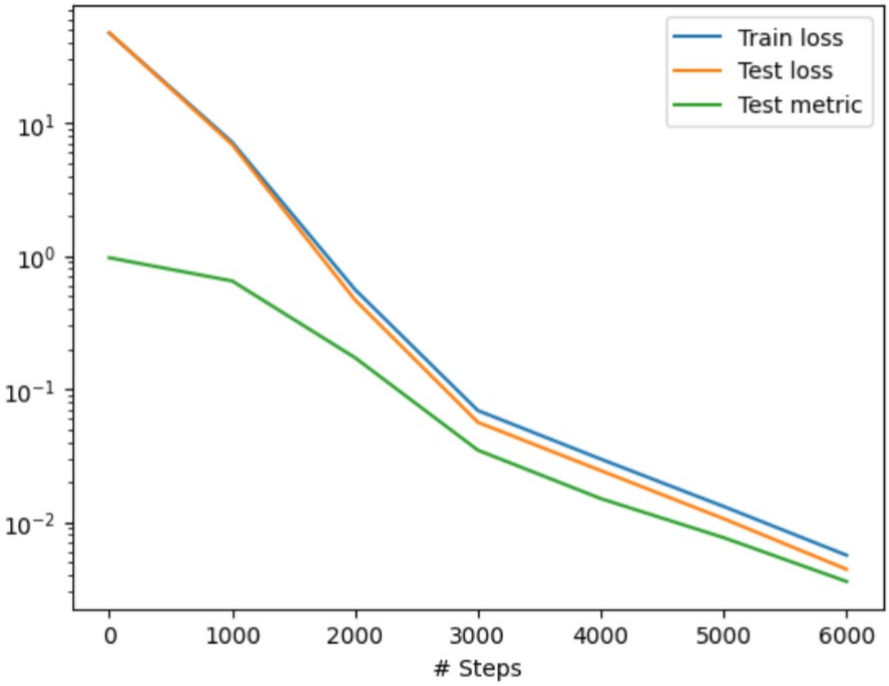


FIGURE 3.2 – Evolution du coût en fonction du nombre d'itérations

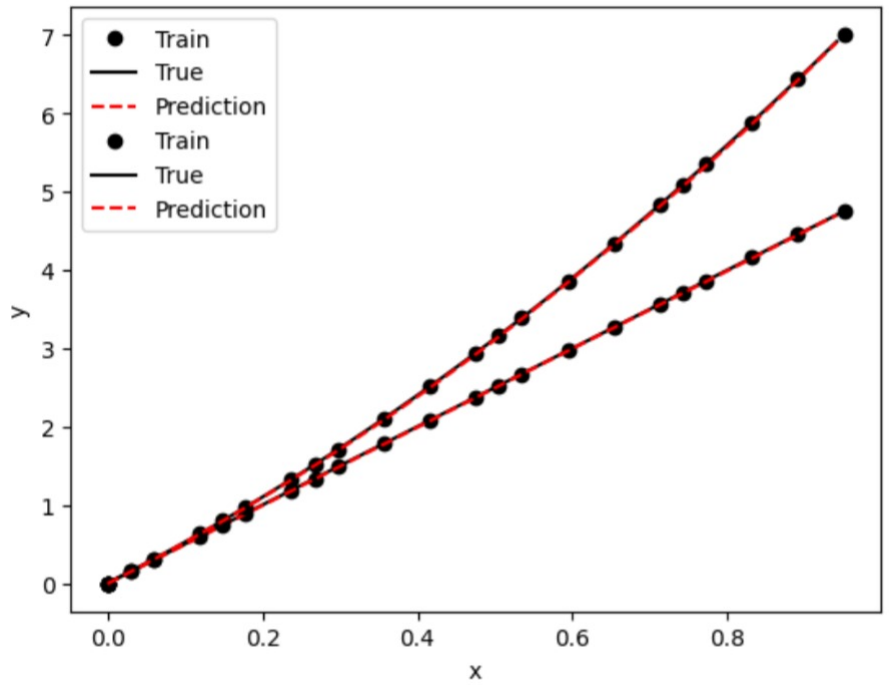


FIGURE 3.3 – Courbes des solutions du système.

Les points sont les points d'entrainement, les courbes en rouge représentent les solutions modélisées et les noires les vraies solutions.

Chapitre 4

L'algorithme quantique H-DES (Hybrid Differential Equation Solver)

Dans ce chapitre nous présentons H-DES (Hybrid Differential Equation Solver), développé par l'entreprise Colibritd pour la résolution des équations différentielles. Cette méthode se base sur le principe du VQA pour mitiger les problèmes des machines quantiques tout en tirant profit des machines classiques d'où la notion *d'hybrid*.

4.1 Fonctionnemen de HDES

Le fonctionnement de HDES est similaire à celui du VQA. Nous avons un ansatz qui va parcourir l'espace des paramètres pour trouver les bons paramètres qui encodent la solution recherchée.
On définit un observable qui sera mesuré à chaque itération, pour évaluer la fonction trouvée aux points d'entrainement. La minimisation du coût permettra de mettre à jour le circuit.

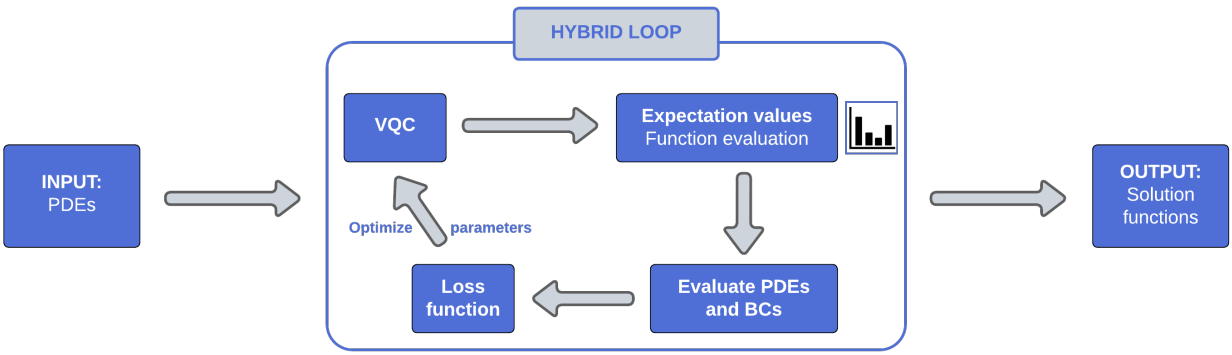


FIGURE 4.1 – Fonctionnement de l'algorithme H-DES

Les étapes de cet algorithmes sont les suivantes :

— On prend en entrée les EDPs (équations aux dérivées partielles) et les conditions aux bords qui y sont associées.
— les équations et les conditions sont encodées avec une fonction coût.
— On a un circuit variationnel quantique qui encode les solutions avec des paramètres.
— On évalue un certain observable par rapport à l'état du circuit précédent puis on utilise cette évaluation pour évaluer les fonctions solutions et le coût.
— On utilise un optimiseur classique pour optimiser le circuit et trouver les paramètres qui minimisent la fonction coût.
— la boucle hybride sera répétée jusqu'à ce qu'une condition sur la tolérance à l'erreur de l'ordinateur quantique soit réalisée

4.1.1 Ansatz et Circuit variationnel quantique utilisés

L'architecture du cicuit utilisé est basée sur la structure d'un Hardware Efficient Ansatz (HEA). Cette architecture est composée d'une couche de rotations sur un qubit à la fois suivi d'une couche de portes CNOT (pour avoir une superposition d'état). Ces couches sont répétées un nombre d fois ; où d est la profondeur du circuit. Plus d est grands plus le nombre de rotations (ainsi le nombre de paramètres) devient grand et l'espace que peut parcourir notre circuit est grand.

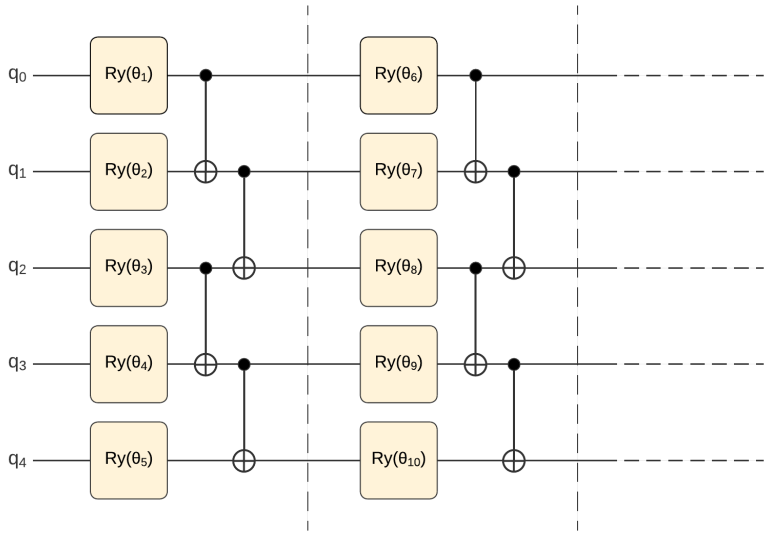


FIGURE 4.2 – Hardware efficient Ansatz

4.1.2 Construction de la fonction solution

Méthode spectrale et polynômes de Chebyshev

L'idée de la méthode spectrale est d'exprimer la fonction solution dans une base de fonctions. La solution de l'équation sera une combinaison linéaire des fonctions de base. Trouver cette solution sera alors de trouver les bons coefficients pour une bonne approximations. Dans H-DES la base de fonction utilisée est celle de Chebyshev. L'expression des polynômes de Chebyshev utilisée est la suivante :

$$\mathrm{Cheb}(k,x) = \begin{cases} \cos(k \arccos x) & \text{if } |x| \leq 1 \\ \cosh(k \operatorname{arcosh} x) & \text{if } x \geq 1 \\ (-1)^k \cosh(k \operatorname{arcosh}(-x)) & \text{if } x \leq -1 \end{cases} \tag{4.1}$$

$x \in \mathbb{R}$ est la variable pour évaluer la fonction et $k \in \mathbb{Z}$ est appelé l'ordre du polynôme de Chebyshev. Une fonction f exprimée dans cette base est de la forme :

$$f(x) = \sum_{k=0}^{C-1} c_k \ \mathrm{Cheb}(k,x) \ , \tag{4.2}$$

avec $c_0, c_1, \ldots, c_{C-1} \in \mathbb{R}$.

Evaluer la solution

Soit u solution de l'équation différentielle donnée, et $|\phi_f\rangle$ un état quantique à n-qubit encondant cette solution.
On a :

$$|\phi_f\rangle = \sum_{i=0}^{2^n-1} a_i |i\rangle \tag{4.3}$$

où les a_i sont les probabilités de réalisation des états $|i\rangle$.

$$\sum_{i=0}^{2^n-1} |a_i|^2 = 1 \tag{4.4}$$

On associe à chaque polynôme de Chebyshev d'un certain ordre un vecteur de base et le coefficient devant ce polynôme la probabilité de mesurer cette base. Comme les probabilités sont positives on peut représenter toutes les fonctions continues possibles. Pour ce faire on réécrit f de la forme suivante :

$$f(x) = \lambda \sum_{i=0}^{2^{n-1}-1} (p_i - p_{i+2^{n-1}}) \ \mathrm{Cheb}(i,x) \ , \tag{4.5}$$

avec $p_i = |a_i|^2$.

Le facteur $(p_i - p_{i+2^{n-1}})$ permet de n'avoir pas que des valeurs positives. Le facteur λ permet à f de parcourir $\mathbb{R}$ sinon elle serait restreinte à $[-1, 1]$ puique $\text{Cheb}(i, x) \in [-1, 1]$. Pour évaluer la fonction f en point on définit un observable $D(x)$ et on écrit

$$f(x) = \lambda \, \langle \psi_f | D(x) | \psi_f \rangle \ , \tag{4.6}$$

avec

$$\begin{aligned} D(x) &= \sigma_z \otimes \left(\sum_{i=0}^{2^{n-1}-1} \text{Cheb}(i, x) ii \right) \\ &= \begin{pmatrix} \text{Cheb}(0,x) & & & & & & \\ & \ddots & & & & & 0 \\ & & \text{Cheb}(2^{n-1}-1,x) & & & & \\ & & & -\text{Cheb}(0,x) & & & \\ 0 & & & & \ddots & & \\ & & & & & -\text{Cheb}(2^{n-1}-1,x) \end{pmatrix} \end{aligned} \tag{4.7}$$

Cette forme permettra d'évaluer f en un point donné au lieu de construire une expression de f de 2^n polynômes de Chebyshev.

Evaluer les dérivées

Pour évaluer le coût, il faut évaluer les dérivées partielles. Les dérivées partielles peuvent être exprimées de la forme :

$$\frac{\partial^q f(x)}{\partial x^q} = \lambda \sum_{i=0}^{2^{n-1}-1} (p_i - p_{i+2^{n-1}}) \, \frac{\partial^q \text{Cheb}(i, x)}{\partial x^q} \ , \tag{4.8}$$

et seront calculées en utilisant le circuit avec l'observable adéquat :

$$\frac{\partial^q f(x)}{\partial x^q} = \lambda \, \langle \psi_f | O_{\partial^q C}(x) | \psi_f \rangle \ , \tag{4.9}$$

4.1.3 Défintion de la fonction coût

La fonction coût encode les EDPs (permet ainsi d'évaluer à chaque itération la valeur prédite et la valeur réelle) et les conditions aux bords.

$$L_{\{\theta\}} = L^{\text{diff}}_{\{\theta\}} + L^{\text{boundaries}}_{\{\theta\}} \tag{4.10}$$

avec

$$L^{\text{diff}}_{\{\theta\}} = \frac{1}{n_s} \sum_{x_s \in S} \left[F[x_s, f_{\{\theta\}}(x_s), \ldots, \partial_x^n f_{\{\theta\}}(x_s)] \right]^2 \tag{4.11}$$

$F[x, f(x), \ldots, \partial_x^n f(x)] = 0$ est l'EDP réecrite en un côté de l'égalité. x_s les points d'entrainement et $\{\theta\}$ les paramètres du circuit. On remarque l'utilisation du MSE (Mean Square Error).

$$L^{\text{boundaries}}_{\{\theta\}} = \eta \frac{1}{n_{BC}} \sum_{x_{BC} \in BC} \left[f_{\{\theta\}}(x_{BC}) - f_{BC}(x_{BC}) \right]^2 \tag{4.12}$$

x_{BC} les points des conditions, n_{BC} le nombre de conditions et η un coefficient pour contrôler les poids. η sera 0 si nous voulons forcer la solution à vérifier les conditions de manière stricte.

Construction d'une fonction pour une vérification stricte des conditions

Cette méthode concerne les conditions sur la fonction f et non ses dérivées et nous permet de faire passer la courbe de notre solution par des points où les conditions doivent être vérifiées sans définir une fonction coût pour ces conditions.

Considérons n_{BC} sur f :

$$\begin{aligned} f(x_0) &= a, \\ f(x_1) &= b, \\ &\vdots \\ f(x_{n_{BC}}) &= k. \end{aligned} \tag{4.13}$$

Soit $\tilde{f}(x)$ la fonction solution au $i$ème iteration, pour chaque point de $\{x_j\}$ on associe la valeur :

$$\text{shift}_j = \tilde{f}(x_j) - f(x_j) \tag{4.14}$$

puis on construit un polynôme d'ordre $n_{BC} - 1$ qui passe par l'ensemble $\{(x_j, \text{shift}_j)\}$

$$\tilde{f}_{\text{shifted}}(x) = \tilde{f}(x) + S(x) \tag{4.15}$$

Les dérivées seront ajustées selon la nouvelle fonction trouvée pour évaluer le coût.

4.1.4 Application

Pour notre application nous étudierons un cas assez simple d'une équation différentielle linéaire, facile à résoudre. Nous utilisons l'algorithme HDES implimenté avec le langage mpqp développé par colibriTD. Ceci est executé comme une simulation en local.

$$\begin{aligned} \text{Équation différentielle :} \quad & \frac{d}{dx}f(x) - f(x) = 0 \\ \text{Condition initiale :} \quad & f(0) = 1 \\ \text{Solution :} \quad & f(x) = e^x \end{aligned}$$

Pour resoudre ce problème, nous avons eu besoin d'un circuit à $n = 3$ qubits avec une profondeur de $d = 2$. Soit au total 6 paramètres. On a fait un total de 50 itérations.
Nous définissons l'observable comme introduite dans la section 4.1.2.

$$\begin{aligned} D(x) &= \sigma_z \otimes \left(\sum_{i=0}^{3} \text{Cheb}(i, x) ii \right) \\ &= \begin{pmatrix} \text{Cheb}(0,x) & & & & & \\ & \ddots & & & & 0 \\ & & \text{Cheb}(3,x) & & & \\ 0 & & & -\text{Cheb}(0,x) & & \\ & & & & \ddots & \\ & & & & & -\text{Cheb}(3,x) \end{pmatrix} \end{aligned}$$

Aussi nous choisissons 20 points pour l'entrainement et 20 points pour le test. L'optimisation des paramètres du circuit se fait avec l'optimiseur BFGS (Broyden–Fletcher–Goldfarb–Shanno)

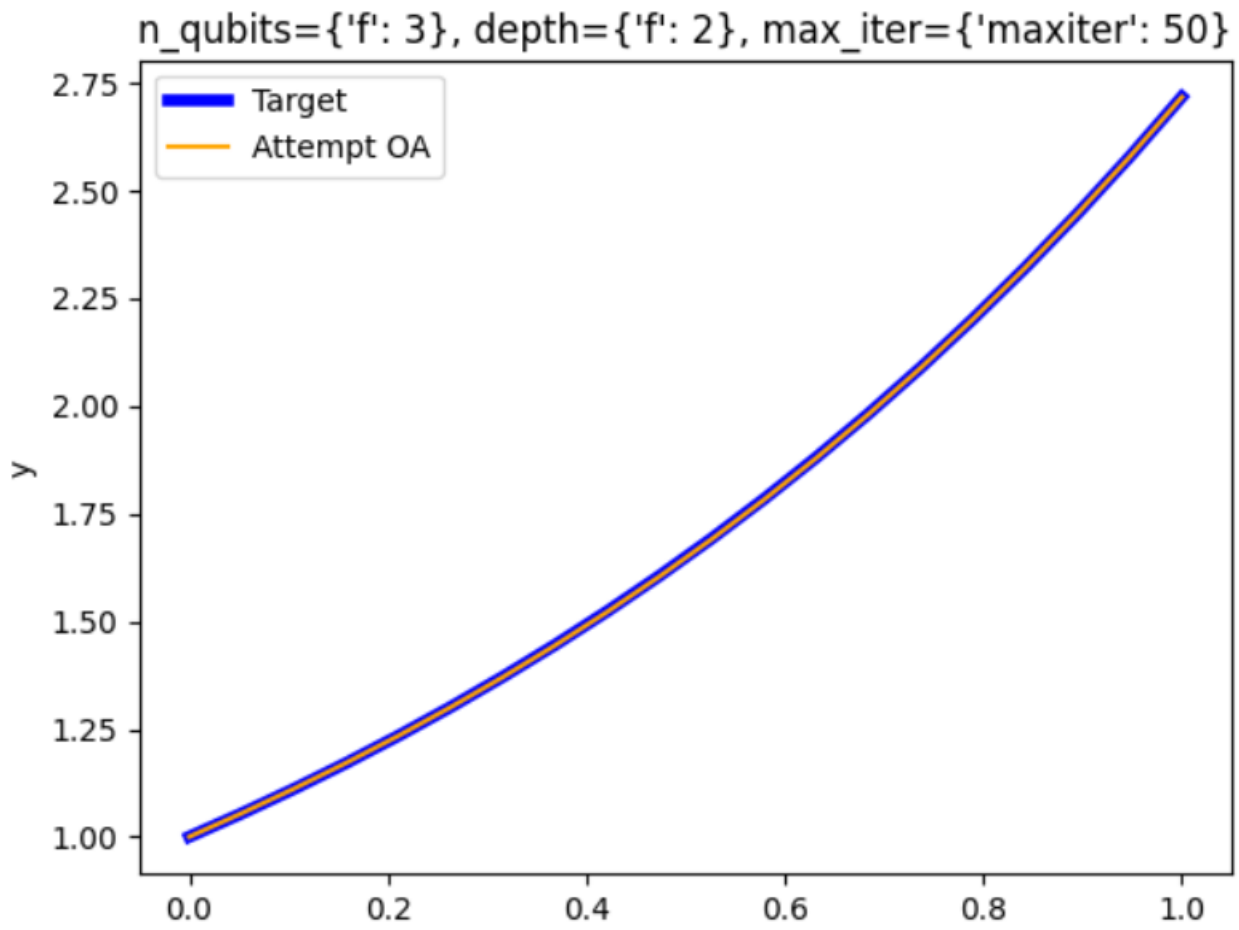


FIGURE 4.3 – Solution pour l'équation différentielle linéaire

Chapitre 5

Comparaison des deux méthodes (PINNs et H-DES) : Cas des équations de la flamme

Une flamme laminaire unidimensionnelle est un type de combustion où le flux de gaz brûlant se déplace de manière uniforme dans une seule direction, souvent caractérisée par un profil de température et de concentration constant dans la direction perpendiculaire au flux. Ce type de modèle est souvent utilisé pour étudier les caractéristiques fondamentales des flammes, telles que la vitesse de flamme, la distribution de température, et les taux de réaction chimique, dans des conditions contrôlées. Il est particulièrement utile pour comprendre les propriétés de base de la combustion sans la complexité des effets multidimensionnels comme la turbulence. Notre système d'équations aux dérivées partielles que nous allons étudier dans ce chapitre à l'aide des PINNs et H-DES est un modèle de flamme unidimensionnelle.

5.1 Résolution de l'équation de la flamme avec les PINNs

$$\begin{cases} m \cdot \frac{\partial f}{\partial x} - \lambda_{Cp\_ratio} \cdot \text{inverse\_Lewis} \cdot \frac{\partial^2 f}{\partial x^2} + A \cdot f \cdot g = 0 \\ m \cdot \frac{\partial g}{\partial x} - \lambda_{Cp\_ratio} \cdot \frac{\partial^2 g}{\partial x^2} - \Delta H \cdot A \cdot f \cdot g = 0 \end{cases} \tag{5.1}$$

où m est une masse, λ_{Cp_ratio} est le rapport entre la conductivité thermique (λ) et la capacité thermique spécifique (C_p), ΔH est l'enthalpie de chaleur, inverse_Lewis est l'inverse du nombre de Lewis (le rapport entre la diffusivité thermique et la diffusivité de masse)

et A un coefficient d'échelle.

$$\text{Conditions initiales :} \quad \begin{cases} f(0) = 1 \\ \left.\frac{\partial f}{\partial x}\right|_{x=1} = 0 \\ g(0) = 1 \\ \left.\frac{\partial g}{\partial x}\right|_{x=1} = 0 \end{cases} \tag{5.2}$$

Les solutions sont :

$$f = \frac{1}{1 + \exp\left(2\beta \cdot (x - 0.5)\right)} - 0.00043 \tag{5.3}$$

$$g = \frac{\delta H + 1}{1 + \exp\left(-2\beta \cdot (x - 0.5)\right)} \tag{5.4}$$

avec $\beta = 7.74$, $\delta H = 4000$, $m = 257$, $\lambda_{Cp\_ratio} = 1$, inverse_Lewis $= 1$, et $A = 1$.

— **Encodage des conditions limites dans des fonctions coûts**

Notre première approche a été de definir une fonction coût pour chaque équation du sytème et pour chaque condition initiale. Les fonctions coûts sont définies comme plus haut (voir la partie PINNs). Malgré les conditions aux bords nous constatons que les solutions trouvées sont triviales (constantes). Ci-dessous nous voyons le résultat de cette première approche.

Pour résoudre ce problème, nous définissons une nouvelle fonction coût de régularisation [8] qui a pour but d'éviter les fonctions triviales.

$$\mathcal{L}_{reg} = \sum_{i=1}^{N} \frac{1}{\hat{u}(\mathbf{x}_i, t_i)^2}$$

où $\hat{u}$ est le modèle produit par le réseau. Chercher le minimum de cette fonction coût permet de trouver les valeurs de u qui ne sont pas des fonctions constantes ou triviales. Malgré cette nouvelle fonction coût, nous n'avons identifié aucun changement majeur dans notre solution.

— **Construction de la solution à partir du modèle**

On constate avec les solutions précédentes trouvées que les modèles ne passent même par les conditions initiales ; même pas de près. Nous avons donc choisi une autre méthode qui consiste à forcer notre solution à vérifier les conditions aux bords de manière stricte. Ainsi, les conditions aux bords ne seront plus encodées dans des fonctions coûts, mais nous allons construire les solutions à partir du modèle à chaque nouvelle itération.

Soit $\hat{u}_\theta$ le modèle produit par notre réseau au $i$ème itération, on construit une

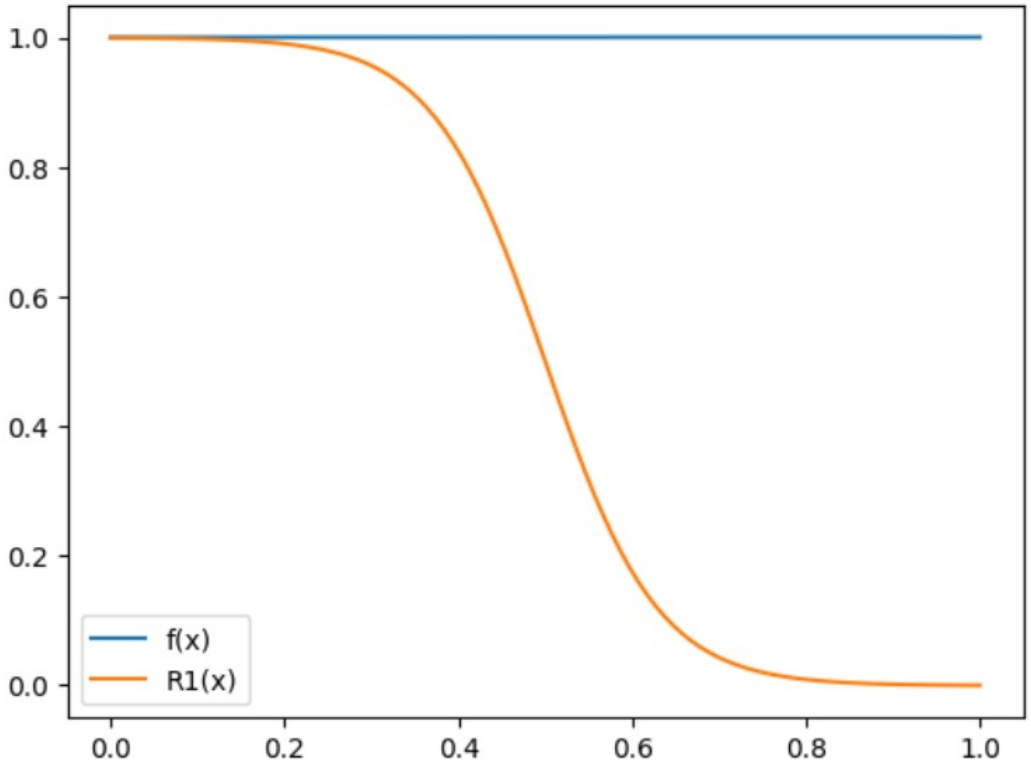


FIGURE 5.1 – Courbe de f (Encodage des conditions dans le coût)

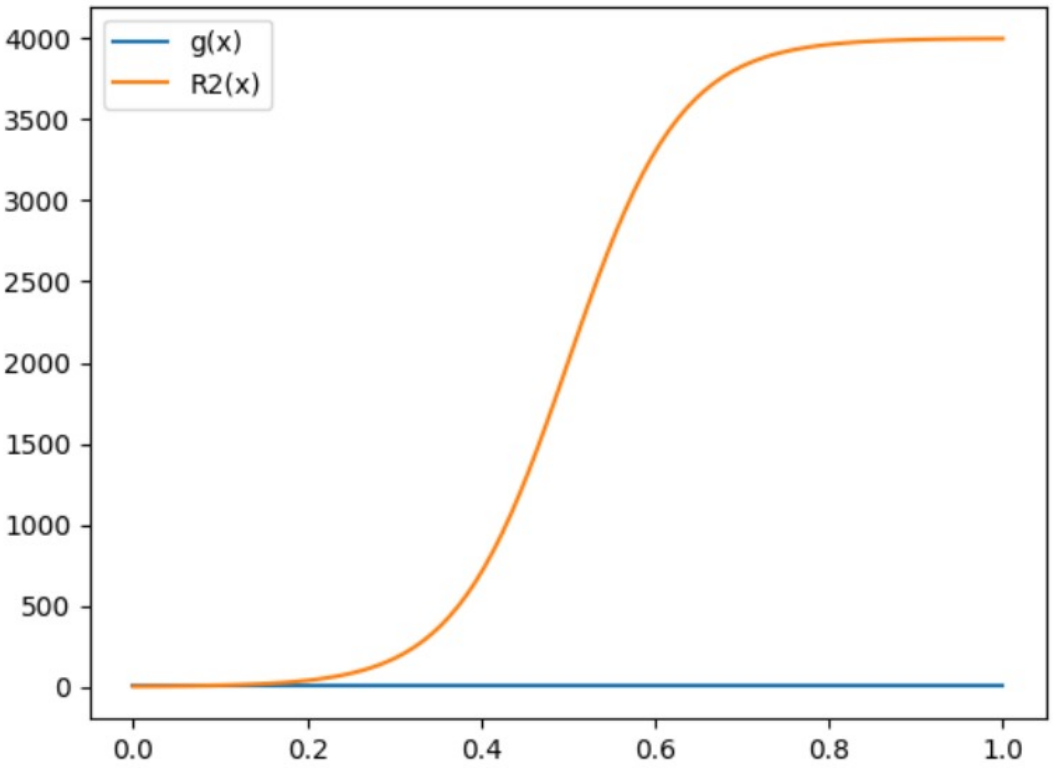


FIGURE 5.2 – Courbe de g (Encodage des conditions dans le coût)

nouvelle fonction $\mathcal{N}(\hat{u}_\theta, z)$ qui sera introduite dans la fonction coût de l'équation différentielle. On ajoute une nouvelle condition sur la valeur $x = 1$.

$$\mathcal{N}(\hat{u}_\theta, z) = x_0(1-x) + x_1 x + x(1-x)\hat{u}_\theta$$

où les conditions sont $u(0) = x_0$ et $u(1) = x_1$. On remplacera x_0 et x_1 par leurs valeurs pour chaque fonction à trouver.

On peut vérifier que la solution vérifie strictement les conditions.
Les courbes ci-dessous montrent les solutions.

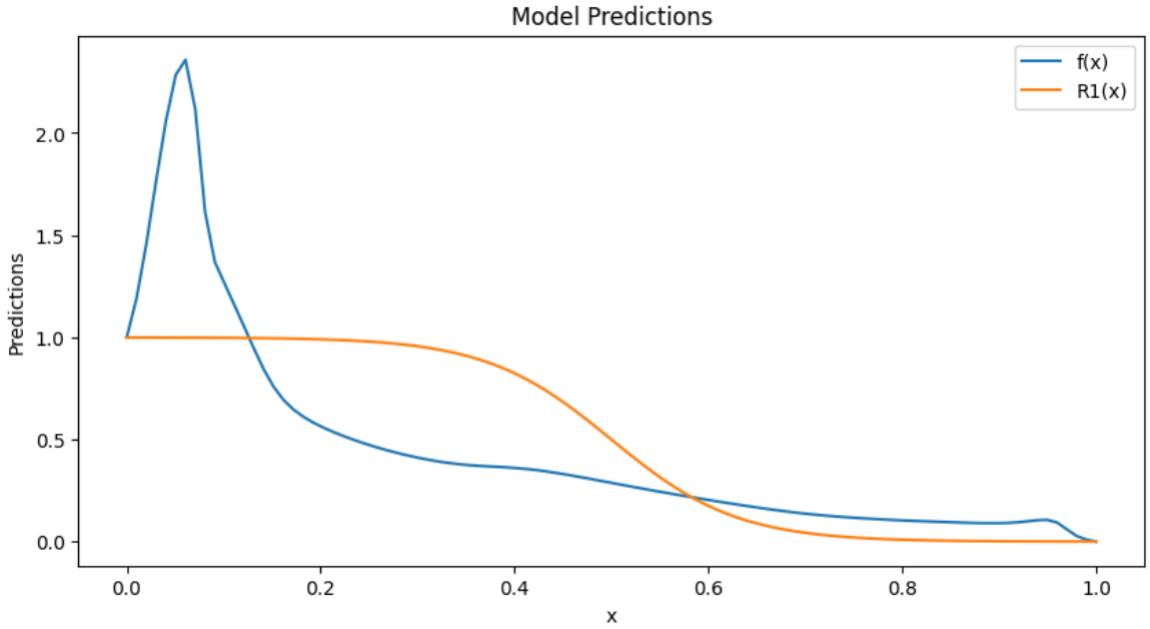


FIGURE 5.3 – Solution avec une condition stricte. Courbe de f

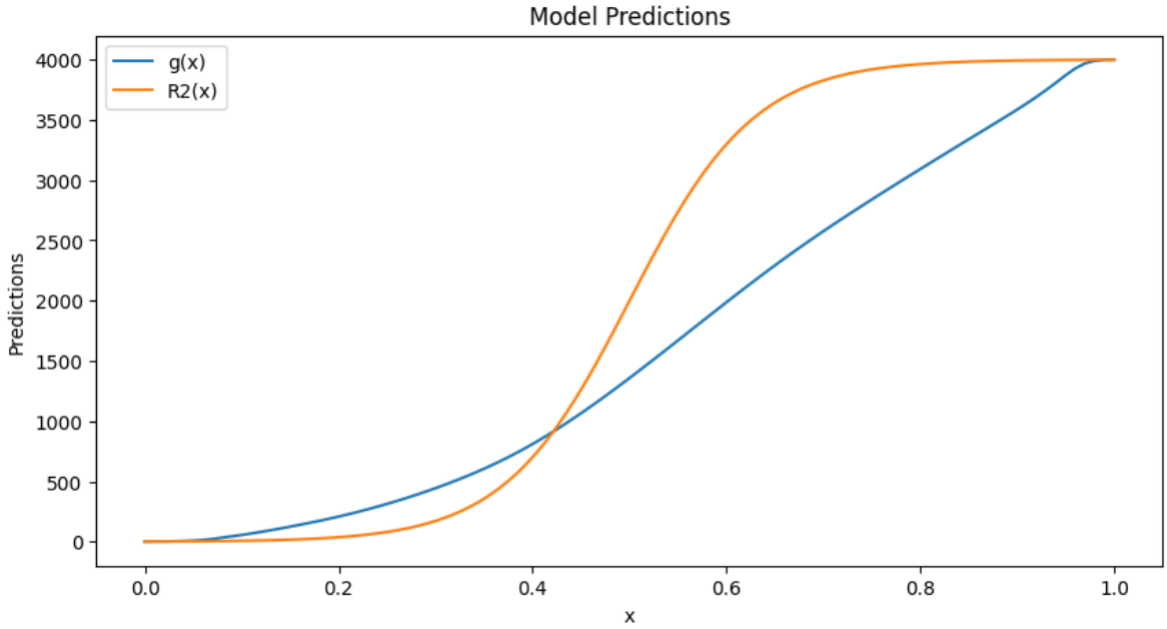


FIGURE 5.4 – solution avec une condition stricte. Courbe de g

Une petite analyse des courbes nous montre que les conditions en 0 et 1 sont vérifiées et les modèles trouvés ne sont pas triviales. Mais nous sommes loin des vraies solutions recherchées.

— **Fonction d'activation adaptative**

Dans les différentes expériences plus haut, nous avons utilisé pour chacune d'elles les fonctions d'activation tanh, sigmoid qui sont les plus souvent utilisées pour les PINNs à cause du fait qu'elles sont différentiables (infiniment différentiable pour tanh) [8].

Après l'échec de ces différentes fonctions d'activation, nous essayons une autre méthode appelée *adaptive activation function* [10] pour la détermination de nouvelles fonctions d'activation non évidente [17] [22] [4] [15]. Cette idée a été suggérée d'abord dans [20].

Nous définissons une fonction d'activation de la forme $G(x) = \sum_{i=0}^{n} \lambda_i \sigma_i(\beta x)$ avec les λ_i des paramètres qui seront mis à jour automatiquement avec le réseau de neurone. $\sigma \in \{sigmoid, tanh, exp\}$. Nous arrivons avec cette méthode à approximer de manière significative nos solutions.

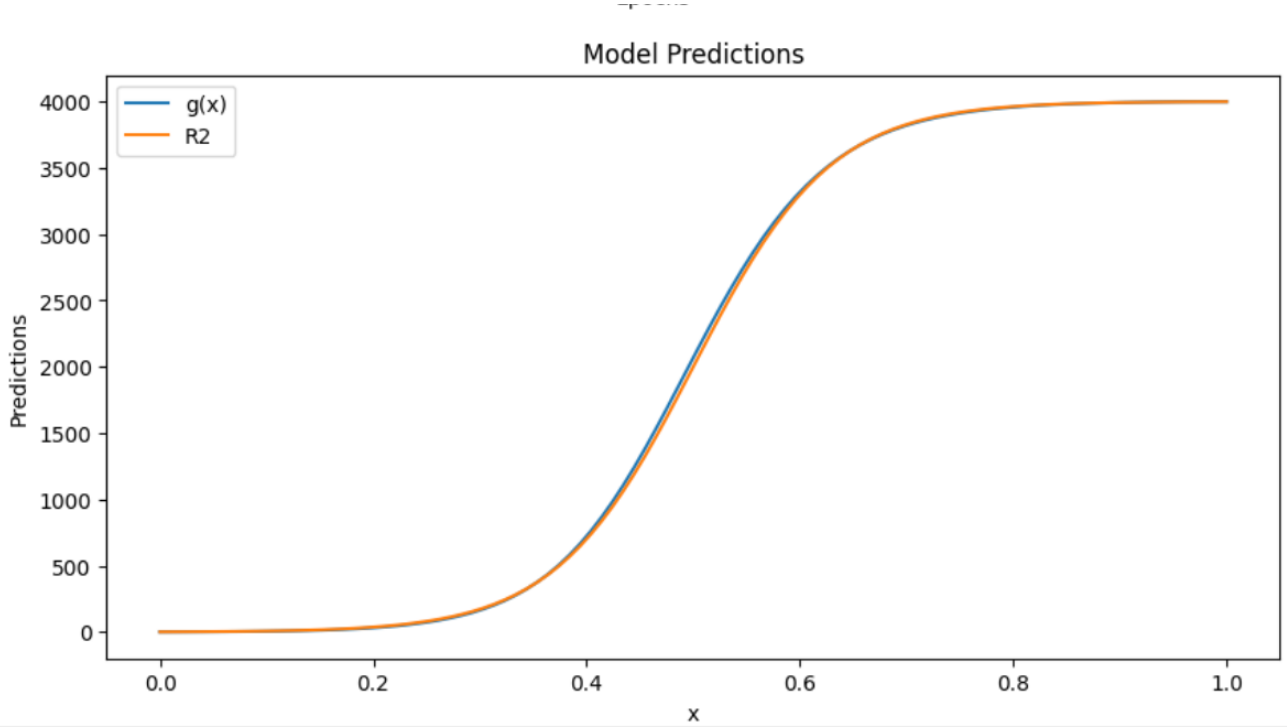


FIGURE 5.5 – courbe de f (Fonction d'activation adaptative).

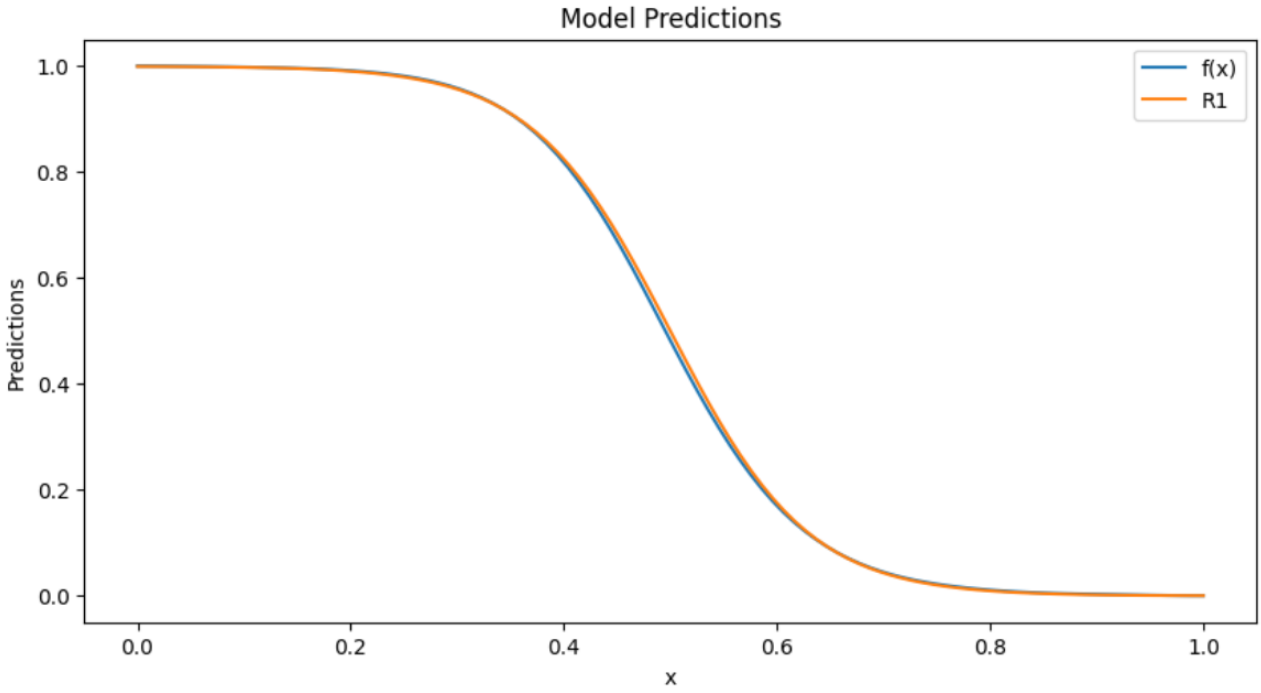


FIGURE 5.6 – courbe de g (Fonction d'activation adaptative).

5.1.1 Etude des erreurs sur l'ensemble de test

Les dernières figures montrent des résultats qui semblent très bien approximer les solutions réelles, mais nous devons prendre en compte la notion d'échelle surtout pour la fonction g. On se décide donc à étudier les erreurs sur les valeurs tests non seulement pour évaluer les modèles, mais aussi essayer de les améliorer.
Pour le test, nous avons choisi 50 points entre $[0, 1]$. Nous calculons les erreurs absolues entre les valeurs prédites et réelles.

$$\text{AE} = |u_i - \hat{u}_i|$$

u_i est la valeur réelle.
$\hat{u}_i$ est la valeur prédite.
Puis nous procédons à la représentation des erreurs en fonction des points de l'intervalle. L'idée est de suivre la distribution des erreurs sur l'intervalle.

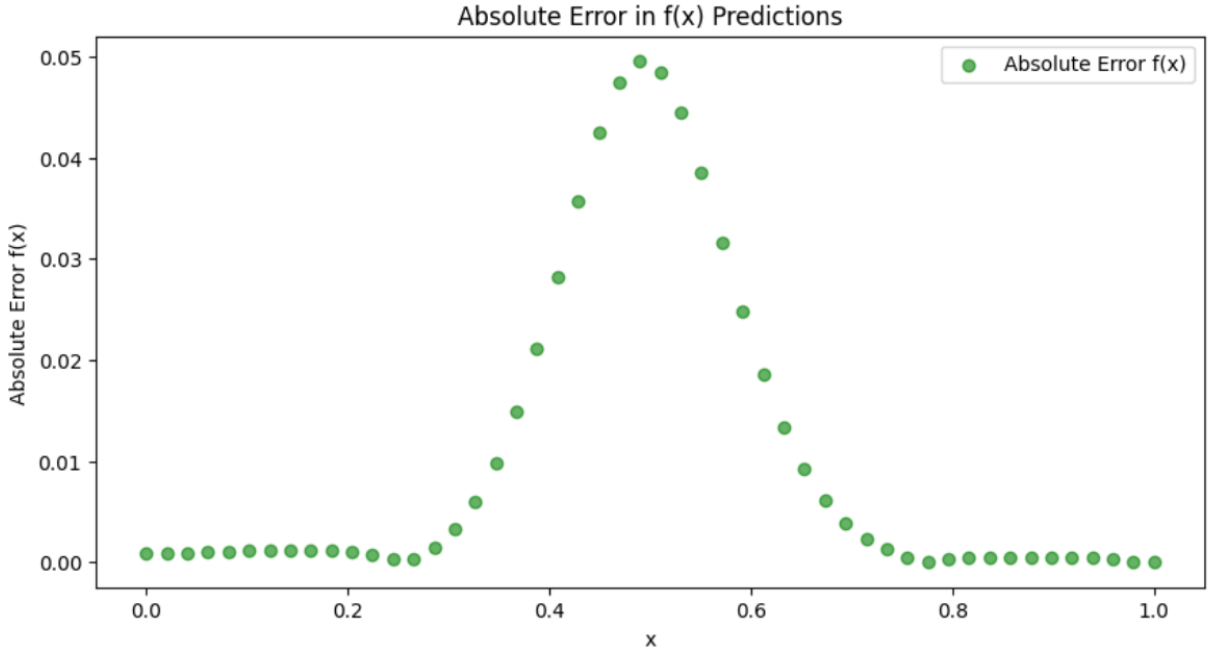


FIGURE 5.7 – distribution des erreurs pour f.

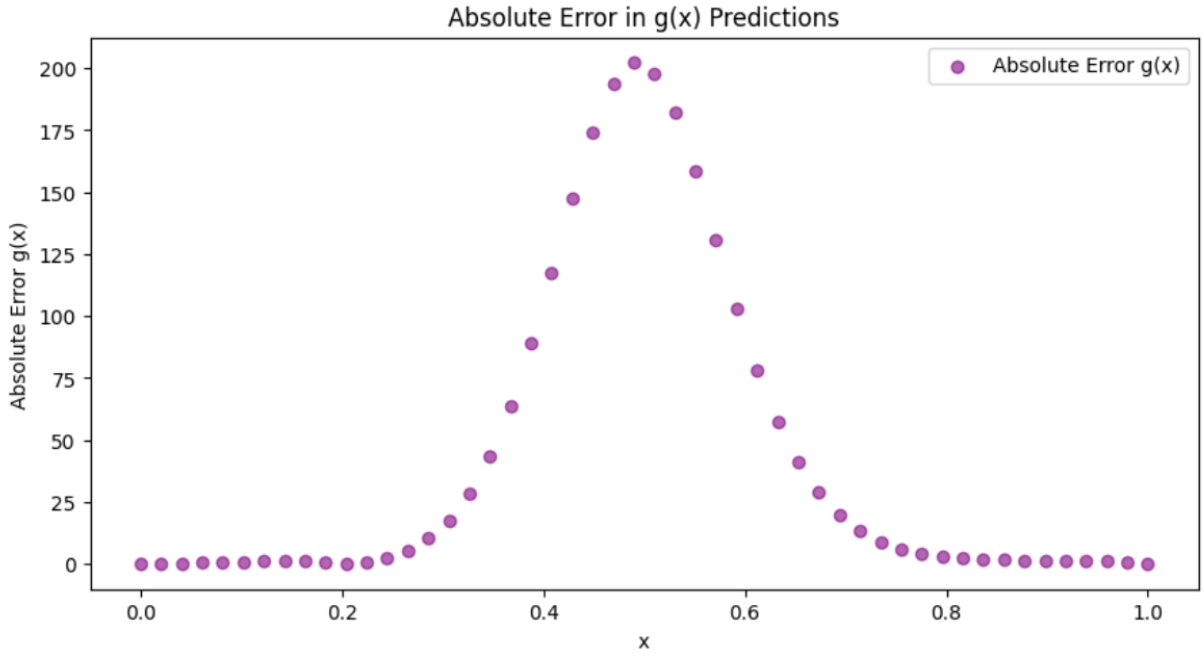


FIGURE 5.8 – distribution des erreurs pour g.

Les valeurs de l'erreur absolue moyenne et la moyenne du carré des erreurs sont :

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |u_i - \hat{u}_i|$$

Pour f MAE $= 0.010405788$
Pour g MAE $= 43.000572$

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

Pour f MSE $= 0.00034999137$
Pour g MSE $= 5947.255$
L'analyse des figures montre que les erreurs suivent une loi normale sur l'intervalle et cette loi a pour centre le centre de l'intervalle. On peut dire plus on va vers le centre de l'intervalle, la prédiction devient moins bonne. Cela peut s'expliquer aussi par les conditions aux bords. Ainsi donc plus on s'approche des bords plus notre modèle est précis.

Ajout d'une condition initiale en 0.5

Une idée pour améliorer le modèle est d'ajouter une condition au centre de l'intervalle. On peut voir les résultats sur les figures suivantes.

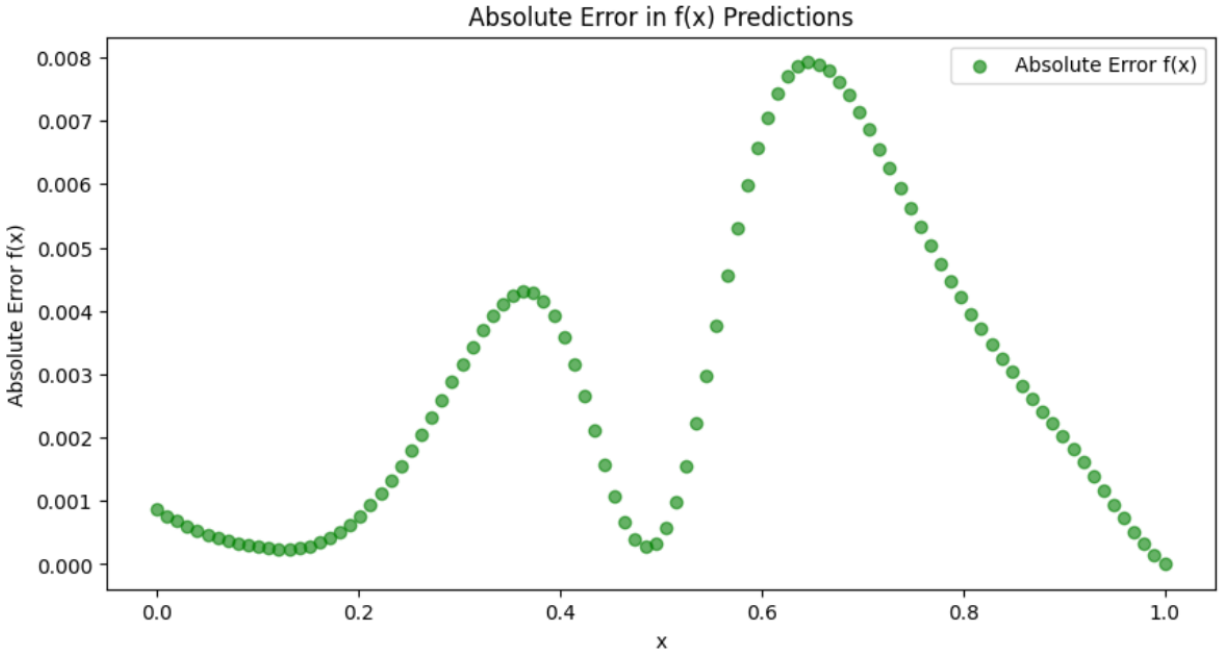


FIGURE 5.9 – Distribution des erreurs pour f avec condition en 0.5.

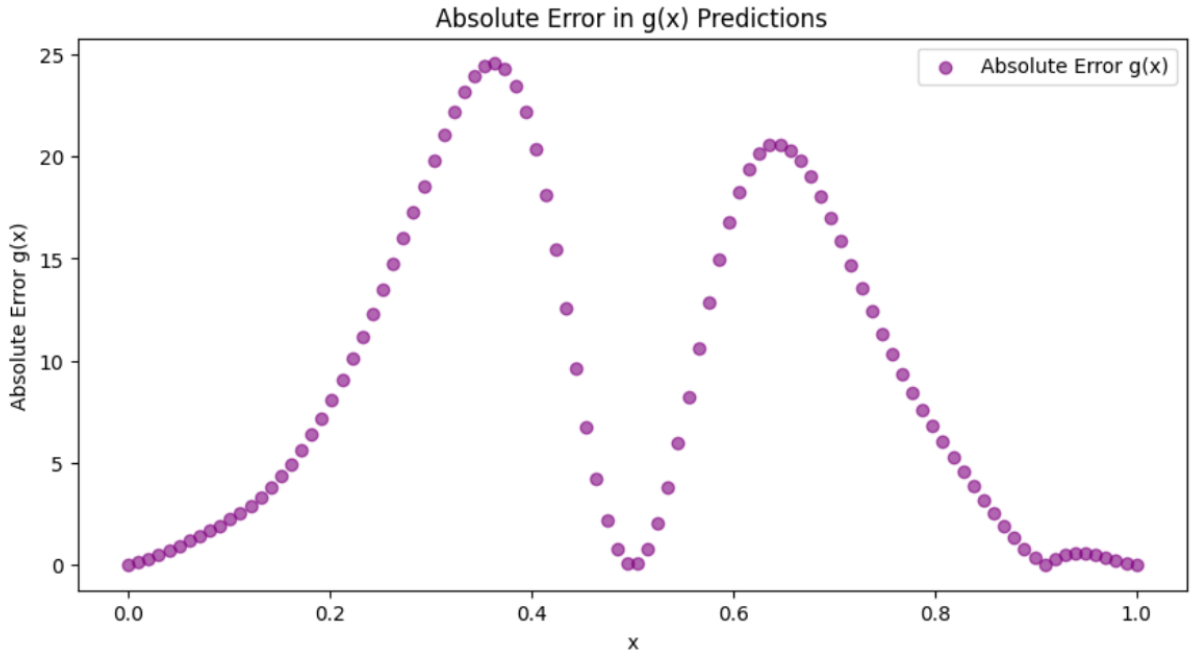


FIGURE 5.10 – Distribution des erreurs pour g avec condition en 0.5.

Les valeurs de l'erreur absolue moyenne et la moyenne du carré des erreurs sont :

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|u_i - \hat{u}_i|$$

Pour f MAE = 0.002847282

Pour g MAE $= 9.150744$

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

Pour f MSE $= 1.3811264e-05$

Pour g MSE $= 146.98663$

On constate une amélioration de l'erreur significativement. Les figures montrent la présence de deux lois normales sur les intervalles $[0, 0.5]$ et $[0.5, 1]$ ceci est dû à la discrétisation de l'intervalle en 2. On pourrait continuer ainsi, mais ça n'aurait aucun sens et notre modèle serait inutile. Nous essayons alors une autre approche.

Choisir un nombre de points d'entraînement plus élevé entre $[0.2, 0.8]$

L'idée ici est d'agir sur le choix des points d'entraînement. Comme les valeurs situées entre $[0.2, 0.8]$ sont mal prédites, nous décidons alors d'augmenter le choix des points d'entraînement entre $[0.2, 0.8]$ tout en laissant le nombre de points d'entraînement invariant dans l'ensemble (même valeur que l'expérience précédente). Nous choisissons alors $\frac{7}{10}$ des points entre $[0.2, 0.8]$ et le reste réparti équitablement entre $[0, 0.2]$ et $[0.8, 1]$. Voici le résultat pour un ensemble de 100 points d'entraînement.

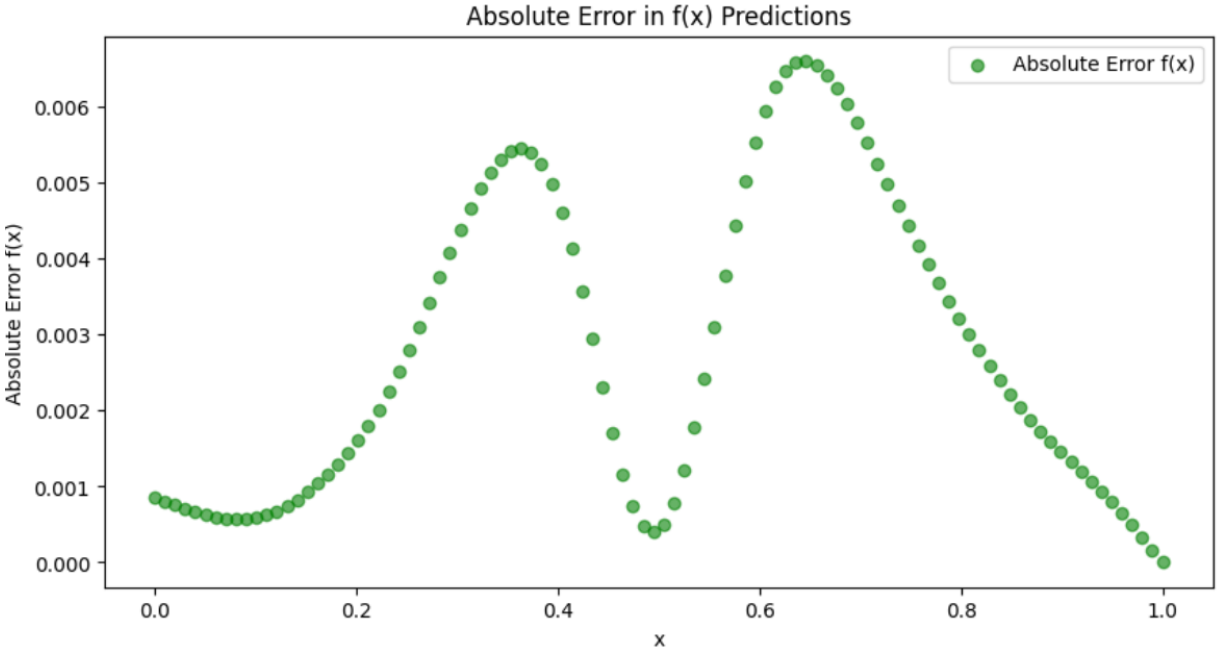


FIGURE 5.11 – Distribution des erreurs pour f avec condition en 0.5.

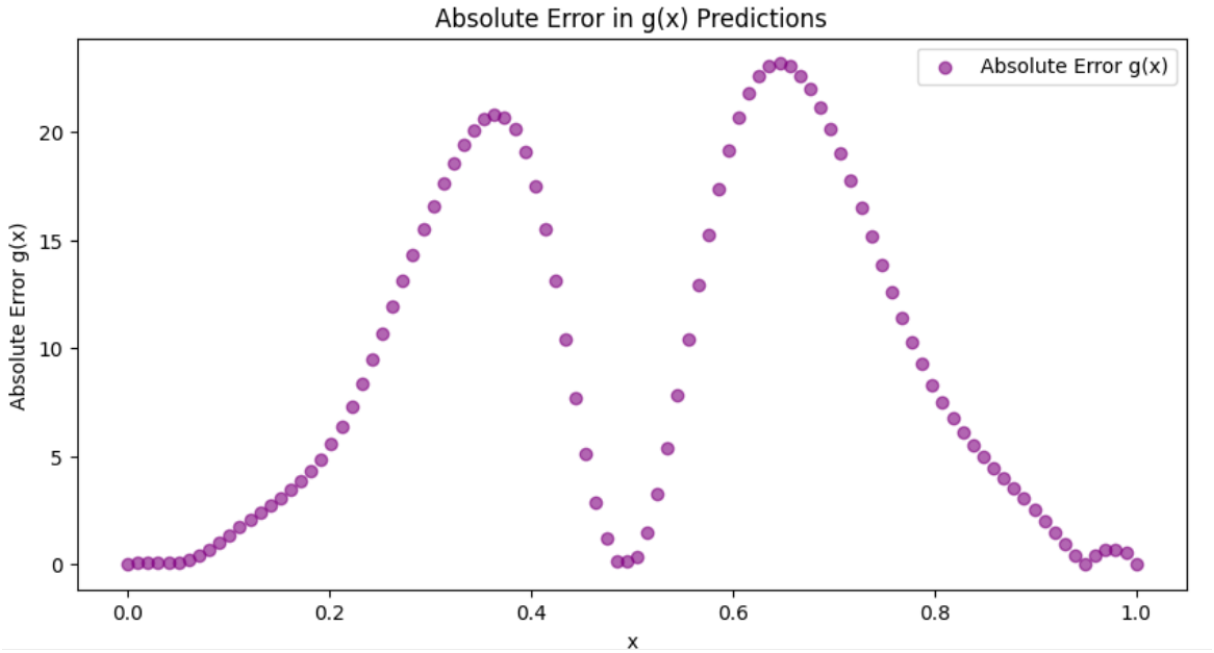


FIGURE 5.12 – Distribution des erreurs pour g avec condition en 0.5.

Les valeurs de l'erreur absolue moyenne et la moyenne du carré des erreurs sont :

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|u_i - \hat{u}_i|$$

Pour f MAE $= 0.0028002132$
Pour g MAE $= 9.05346$

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

Pour f MSE $= 1.1832898e - 05$
Pour g MSE $= 142.59273$

On constate une amélioration significative de l'erreur surtout pour la fonction g. On peut se poser la question si cette solution ne peut être améliorée au niveau des hyper-paramètres du réseau. Pour ce faire nous nous sommes pencher sur différents aspects de notre solution.

— **Initialisation des poids :** Pour assurer une bonne convergence et éviter les phénomènes comme l'explosion du gradient (des gradients trop grands, on parle d'exploding gradient) ou la disparition du gradient (Des gradients trop petits ; on parle de vanishing gradient). Nous avons utilisé l'initialisation avec xavier (ou Glorot

initialisation) pour initialiser les poids et les biais [13].

— **Fonction d'activation :** Nous avons vu que notre fonction d'activation est une combinaison de fonction d'activation. Notre idée est de voir laquelle à plus d'impact et ainsi diminuer le nombre de paramètres. Pour cela, nous avons essayé de voir les poids pour chaque fonction au niveau de notre solution. Mais les poids ne semblent pas pencher vers une fonction en particulier. Nous avons tout de même fait l'expérience en éliminant les fonctions *Sigmoid* et *Tanh*. On a constaté que les solutions approximent bien les vraies fonctions. L'utilisation de la fonction *exponentiel* seul sans paramètres n'a pas donné une bonne approximation.

— **Nombre de points d'entraînement :** Après plusieurs essais, nous avons obtenu une bonne approximation pour 100 points d'entraînements et 50 points test. En dessous de 50 points test on constate que les fonctions ne sont pas bien lisses.

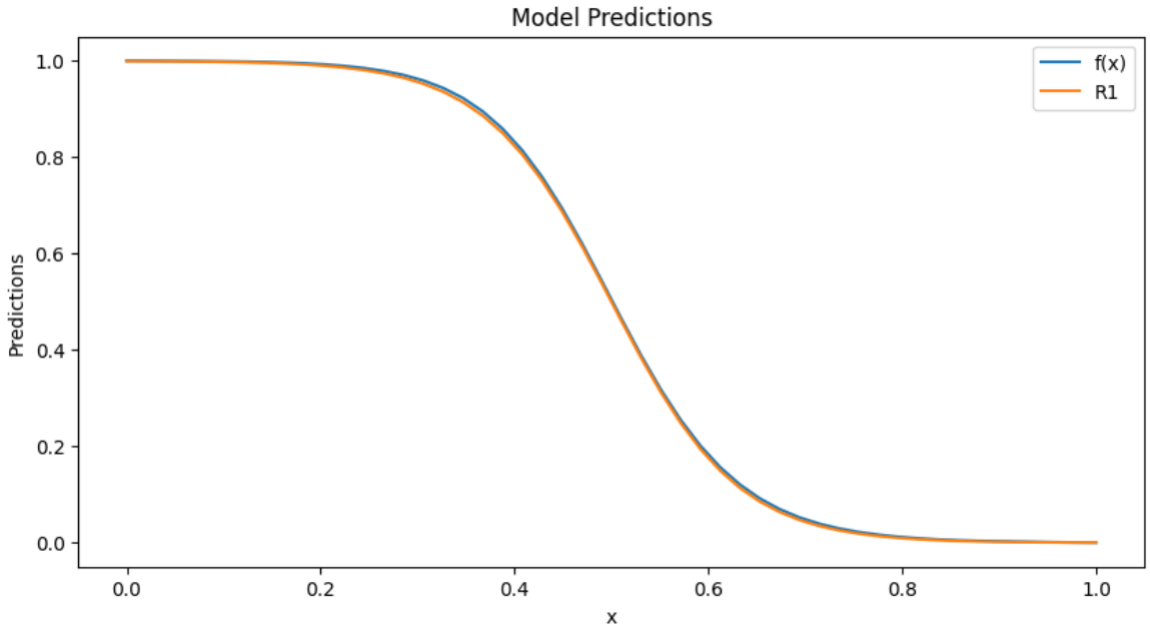


FIGURE 5.13 – Solution finale f.

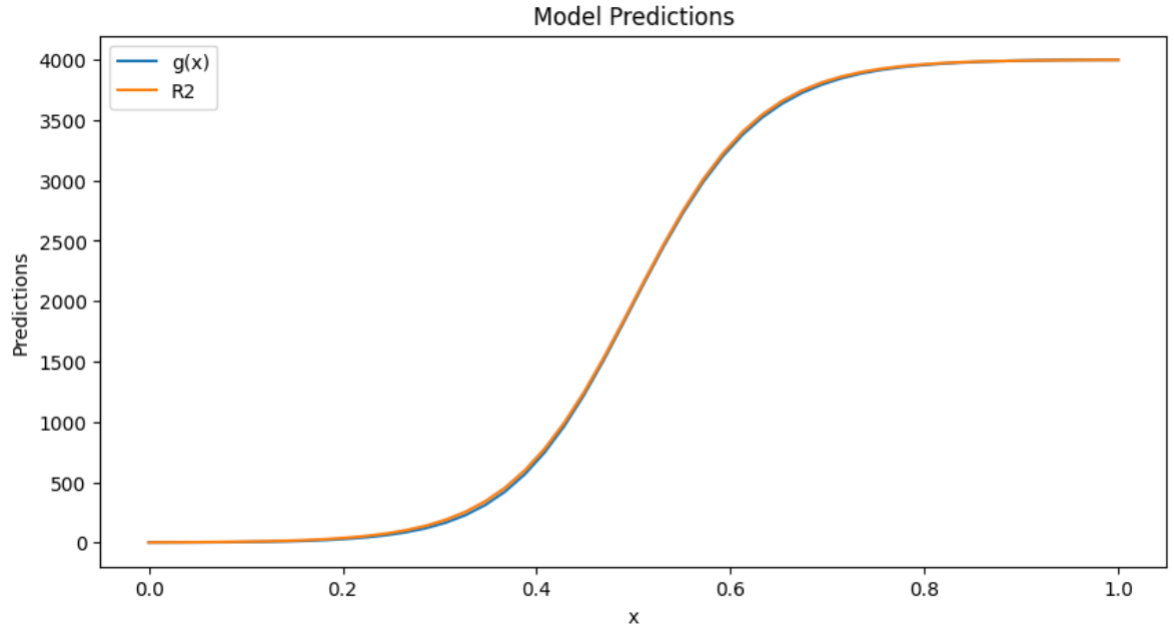


FIGURE 5.14 – Solution finale g.

Remarque. *Les expériences précédentes ont été réalisées avec* ***Pytorch*** *et un* ***GPU NVIDIA GeForce RTX 3060****.*

5.2 Résolution de l'équation de la Flamme avec H-DES

Nous rappelons notre équation :

$$\begin{cases} m \cdot \frac{\partial f}{\partial x} - \lambda_{Cp\_ratio} \cdot \text{inverse\_Lewis} \cdot \frac{\partial^2 f}{\partial x^2} + A \cdot g \cdot f = 0 \\ m \cdot \frac{\partial g}{\partial x} - \lambda_{Cp\_ratio} \cdot \frac{\partial^2 g}{\partial x^2} - \Delta H \cdot A \cdot g \cdot f = 0 \end{cases} \tag{5.5}$$

où m est une masse, λ_{Cp_ratio} est le rapport entre la conductivité thermique (λ) et la capacité thermique spécifique (C_p), ΔH est l'enthalpie de chaleur, inverse_Lewis est l'inverse du nombre de Lewis (le rapport entre la diffusivité thermique et la diffusivité de masse) et A un coefficient d'échelle.

$$\text{Conditions initiales :} \quad \begin{cases} f(0) = 1 \\ \left.\frac{\partial f}{\partial x}\right|_{x=1} = 0 \\ g(0) = 1 \\ \left.\frac{\partial g}{\partial x}\right|_{x=1} = 0 \end{cases} \tag{5.6}$$

avec $\beta = 7.74$, $\delta H = 4000$, $m = 257$, $\lambda_{Cp\_ratio} = 1$, inverse_Lewis $= 1$, et $A = 1$.

Nous commençons notre résolution par définir le nombre de qubit $n = 3$ et $d = 2$ et nous faisons 50 itérations.

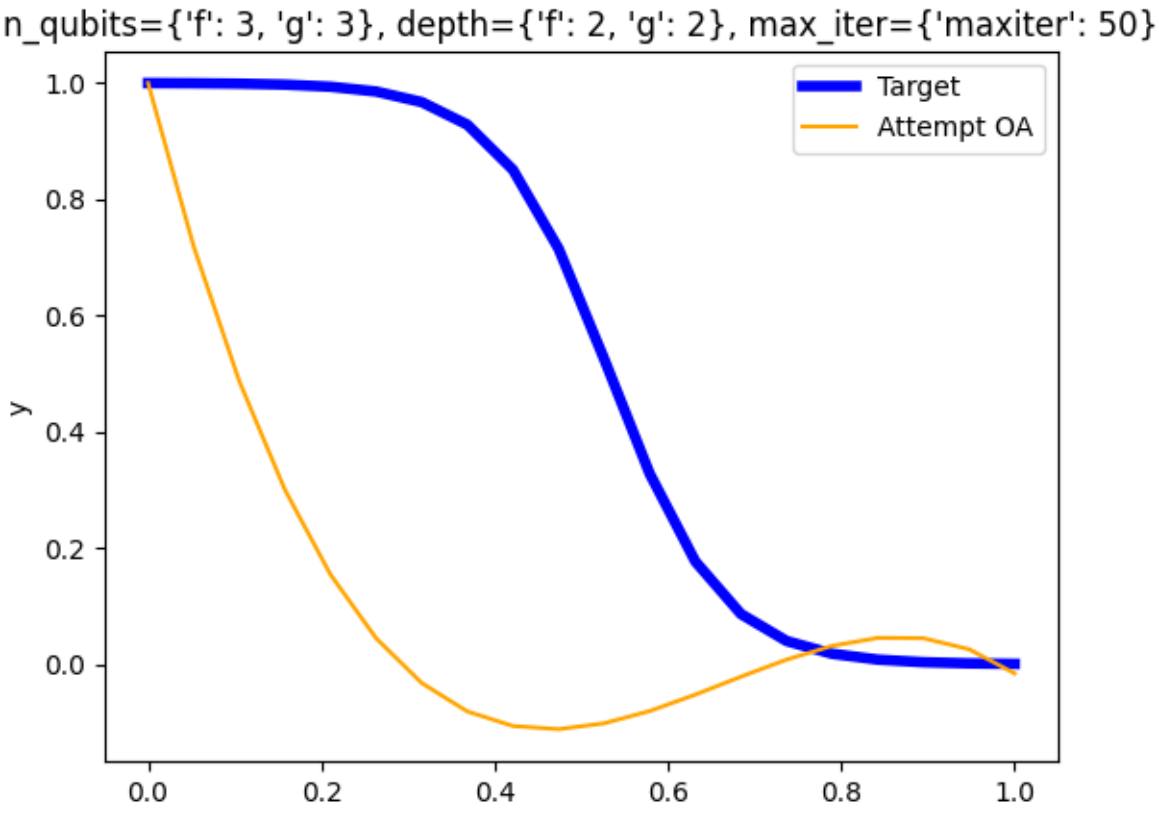


FIGURE 5.15 – courbe f avec n = 3, d = 2.

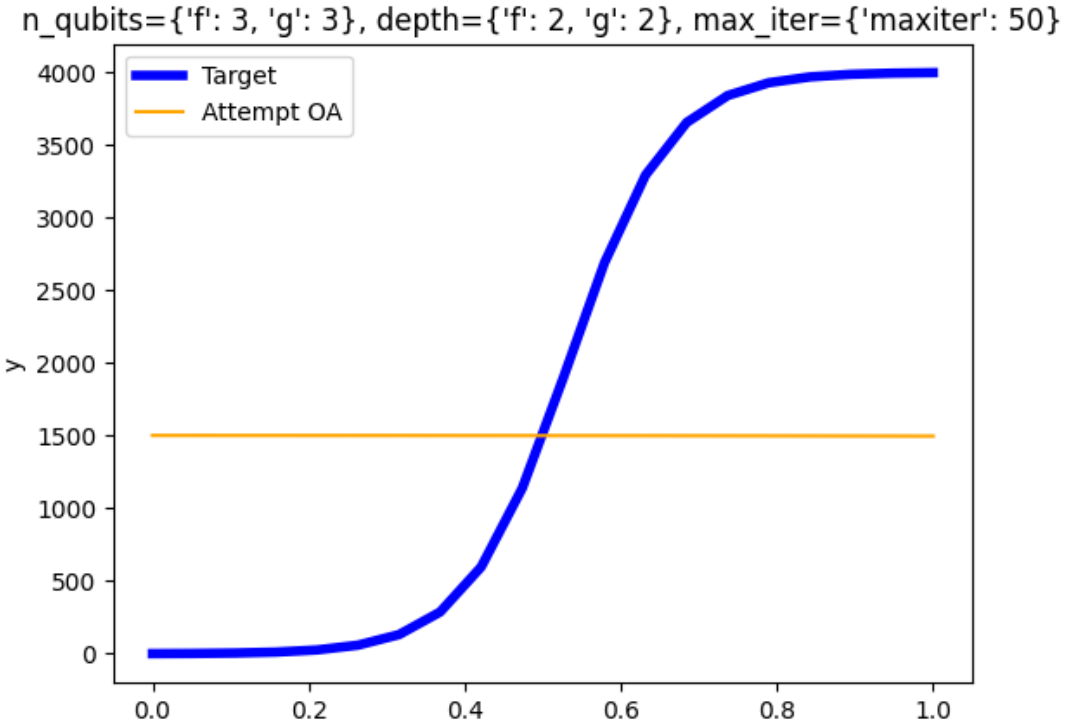


FIGURE 5.16 – courbe g avec n = 3, d = 2.

Les figures montrent des résultats qui sont bien loin des solutions. Nous augmentons d'abord le nombre d'itérations à 100 pour voir l'évolution des résultats. On remarquera que le nombre d'itérations que nous choisissons ici est très bas comparé au cas des PINNs. Ceci est dû est au fait que les itérations sont coûteuses en matière de temps d'exécution (pour chaque itération, il faudra exécuter une partie quantique avec le calcul des dérivées pour chaque fonction chebyshev et une partie classique.) et d'erreurs pour le circuit. Notre idée est alors de faire des essais préalables pour déterminer le nombre qubit et de profondeur du circuit adéquat puis après essayer d'optimiser le résultat en augmentant le nombre d'itérations.

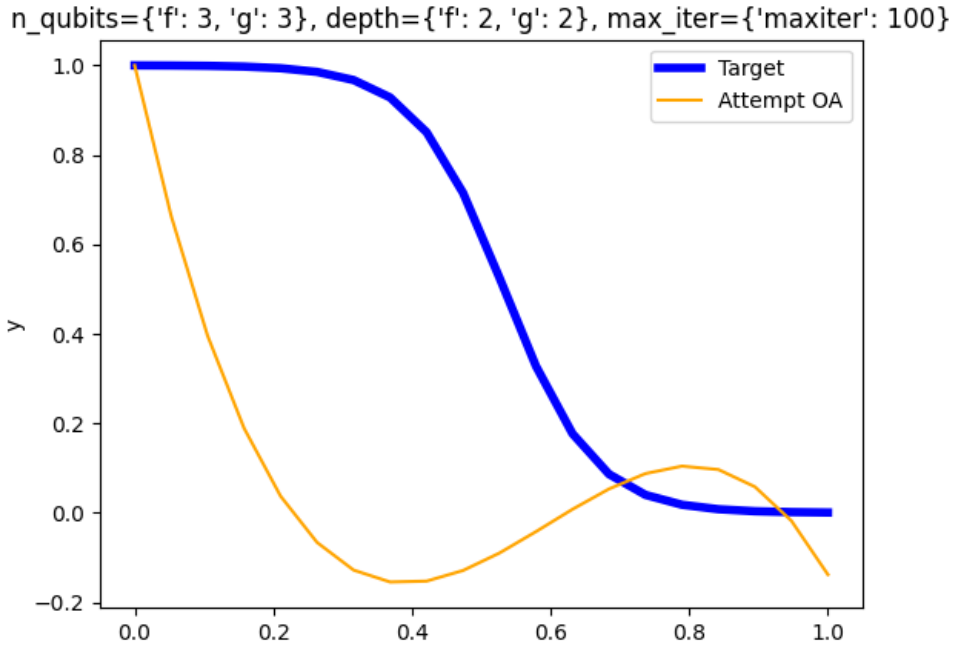


FIGURE 5.17 – courbe f avec $n = 3$, d $= 2$ avec 100 itérations.

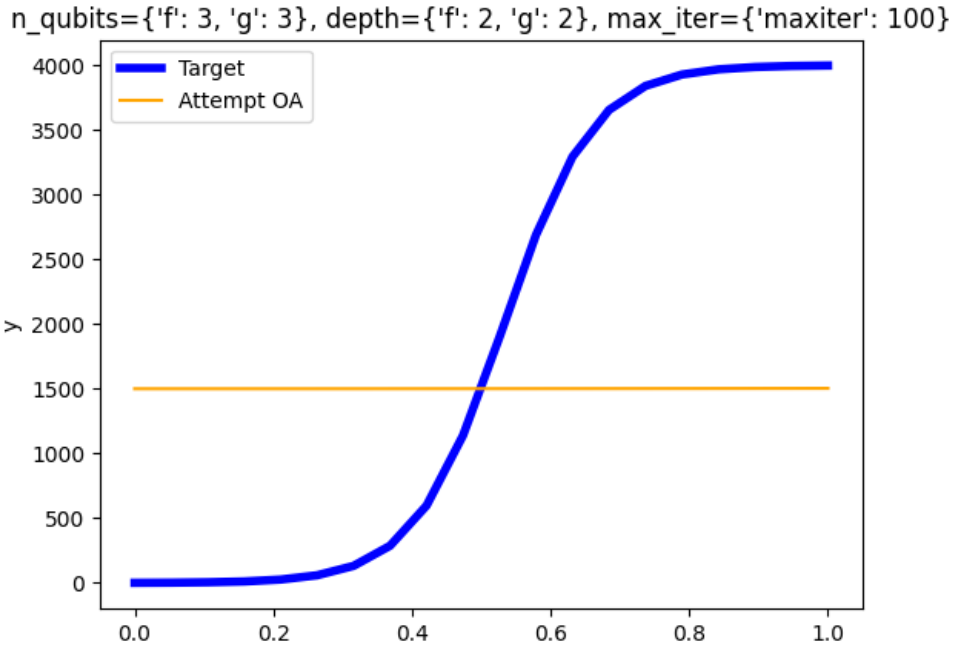


FIGURE 5.18 – courbe g avec $n = 3$, d $= 2$ avec 100 itérations.

On ne constate pas une évolution des résultats après avoir doublé le nombre d'itérations. Nous avons alors augmenté la profondeur du circuit à $d = 5$. Les résultats nous montrent clairement que le nombre $n = 3$ est insuffisant pour représenter les solutions de notre système.

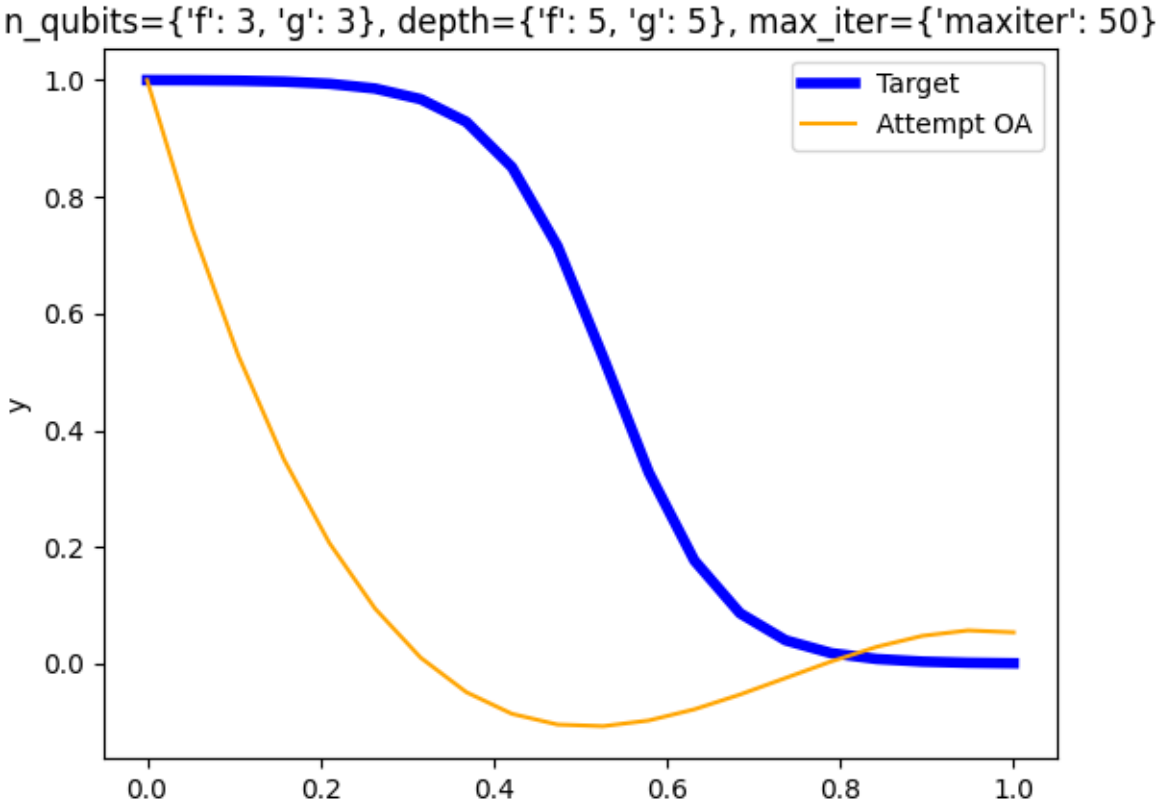


FIGURE 5.19 – courbe f avec $n = 3$, d $= 5$ avec 100 itérations.

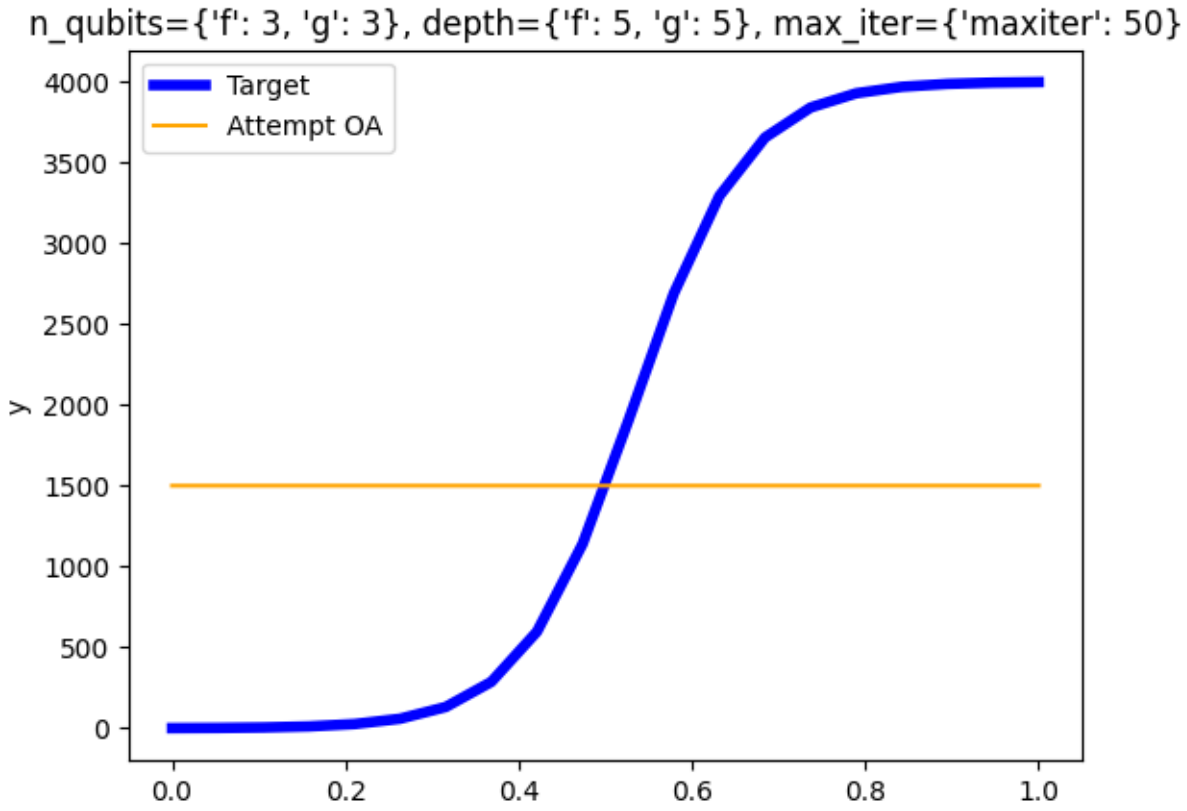


FIGURE 5.20 – courbe g avec $n = 3$, d $= 5$ avec 100 itérations.

Nous augmentons le nombre de qubit $n = 5$ et la profondeur du circuit $d = 6$. Nous avons choisi pour optimiseur **BFGS** avec 1500 itérations pour trouver notre résultat ci-après. Pour une meilleure précision, l'on peut augmenter le nombre d'itérations. Pour des raisons de temps d'exécution trop long, nous avons choisi de nous arrêter à 1500. Nous avons d'abord entraîné le modèle sur $[0, 0.5]$ et avec le modèle obtenu nous avons continué l'entraînement sur $[0.5, 1]$.

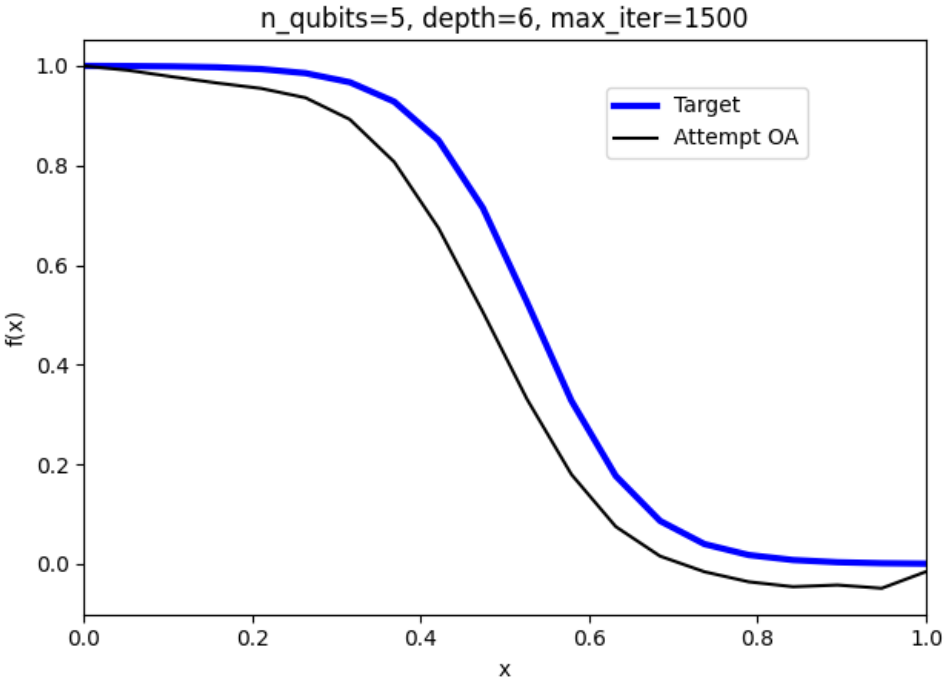


FIGURE 5.21 – courbe de f avec H-DES.

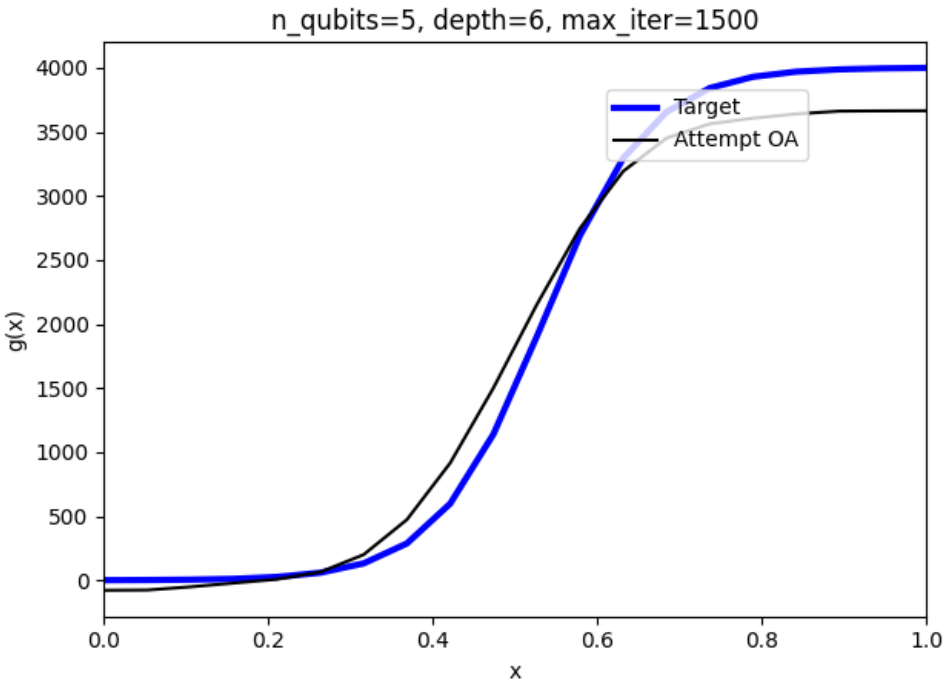


FIGURE 5.22 – courbe de g avec H-DES.

Remarque. *Les expériences ont été réalisées avec* ***QLM*** *sur le* ***QPU*** *(Quantum Proces-*

*sing Unit) d'**ATOS**.*

5.3 Comparaison des deux méthodes cas de l'équation de la flamme

Les deux méthodes de résolution des équations différentielles ont de nombreuses point communs. Parmi ces points communs, on peut citer :

Un ensemble de paramètres

Les 2 méthodes ne parcourent pas les mêmes espaces. Pour notre méthode avec les PINNs, nous avons les paramètres qui sont les poids et les biais. Spécifiquement, pour la solution trouvée ici pour la flamme, on a :

- Une couche d'entrée à une seule entrée composée de 10 neurones. On a ici 10 poids +10 biais soit 20 paramètres.
- Trois couches cachées composées chacune de 10 neurones. On a pour les couches cachées 200 poids et 20 biais soit 220 paramètres.
- Une couche de sortie avec 2 sorties pour chacune des deux fonctions prédictes. On a ici 20 poids et 2 biais soit 22 paramètres.

Au total pour notre réseau de neurones, nous avons 268 paramètres (en ajoutant les paramètres pour la fonction d'activation).
Pour circuit variationnel du HDES nous avons $5 * 6 = 30$ paramètres (nombre de qubit fois la proofondeur du circuit). On constate que le nombre de paramètres du PINN est largement supérieur à celui de HDES.
On peut voir avec les figures ci-dessous un modèle avec 30 paramètres.

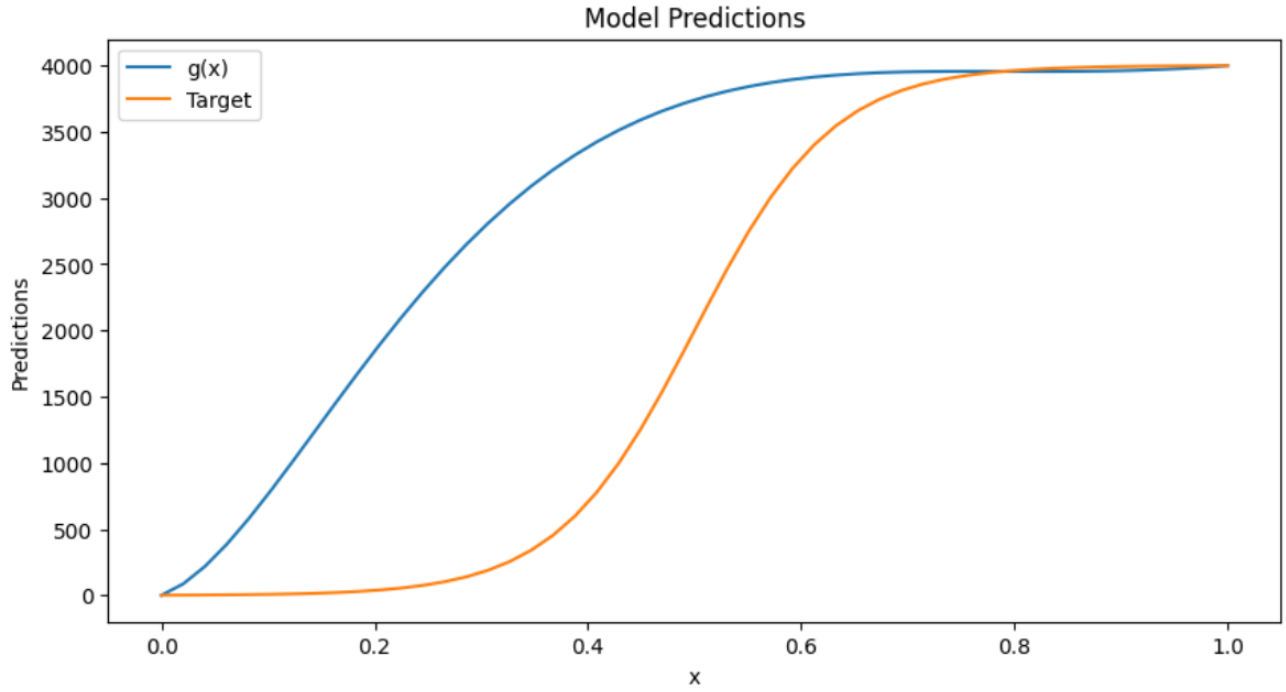


FIGURE 5.23 – Courbe de f avec 30 paramètres

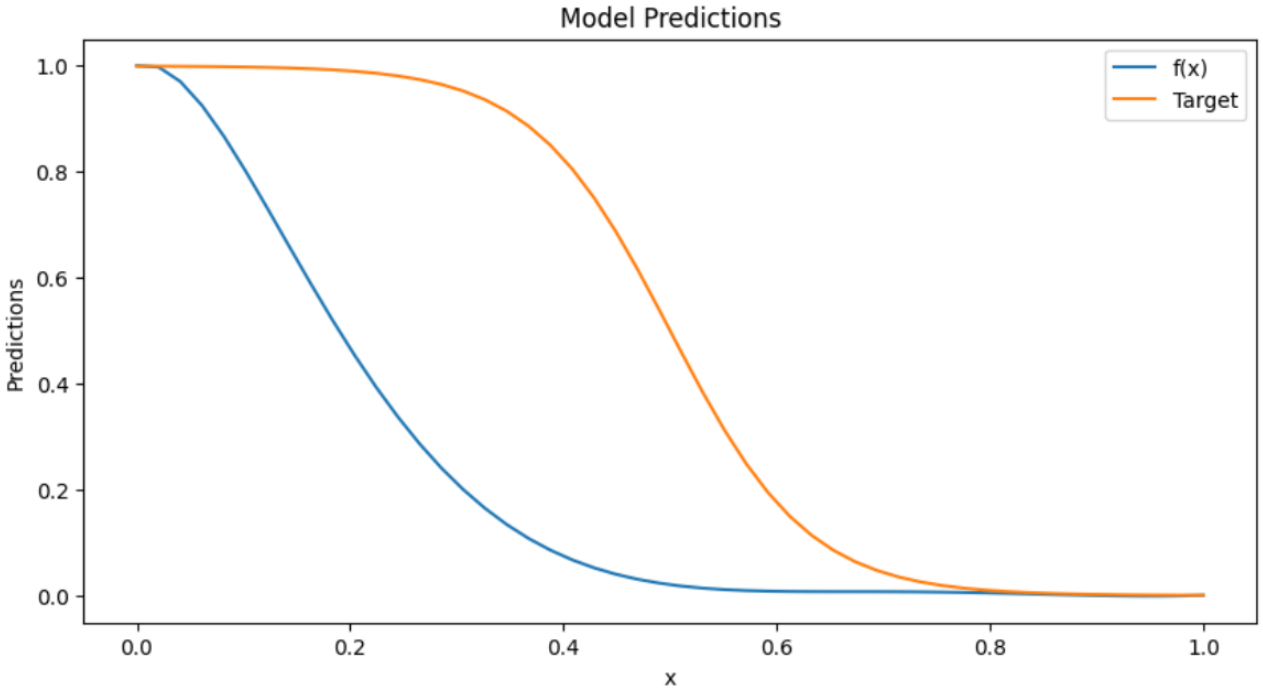


FIGURE 5.24 – Courbe de g avec 30 paramètres

L'optimiseur

L'optimiseur utilisé pour la solution avec les PINNs est l'optimiseur Adam. Nous avons essayé l'optimiseur BFGS (Broyden–Fletcher–Goldfarb–Shanno) mais nous avons constaté des fois que notre fonction trouvée est une fonction en escalier. Nous avons une meilleure prédiction avec Adam. Au niveau niveau HDES, l'optimiseur utilisé est BFGS.

Le Nombre d'itérations

Nous faisons au total 75000 epochs pour l'optimisation avec les PINNs (5000 avec un pas 0.01 pour la descente de gradient et 25000 avec 0.001). Pour le H-DES nous avons eu besoin 1500 itérations pour le résultat obtenu, mais on conseille au moins 2000 itérations pour une meilleure approximation.

La fonction coût

Les deux méthodes ont utilisé une méthode similaire pour la déclaration. En effet pour chacune des 2 méthodes, nous avons une fonction coût pour les équations et une construction d'une nouvelle fonction à partir du modèle (à chaque itération) dans le but d'avoir une stricte vérification des conditions initiales. On peut constater qu'une des solutions se trouve sur une échelle de 0 à 4000 pour éviter que le MSE ne soit trop grand nous avons choisi un poids de $\frac{1}{4000^2}$ pour la fonction coût au niveau du PINN.

Nous établissons ci-dessous un tabeau comparatif.

Points de comparaison	**PINNs**	**HDES**
Les paramètres	268	30
Métrique pour le coût	MSE	MSE
Optimiseur	Adam	BFGS
Itérations	75000	1500

TABLE 5.1 – Tableau comparatif des 2 solutions PINNs et HDES.

La comparaison des PINNs et de HDES sur le problème de la flamme laminaire montre que sur ce problème, les PINNs ont besoin de plus de paramètres pour avoir un résultat précis. Cela peut s'expliquer par le fait que la dimension de l'espace parcouru par le circuit quantique de HDES ($D = 2^n$, soit $2^{\cdot 5} = 32$ dans notre cas) est plus grande que celle de l'espace du réseau de neurones. La présence de plus de paramètres au niveau des PINNs peut amener un long temps nécessaire à l'optimisation de ces paramètres par l'optimiseur. On constate aussi que le nombre d'itérations est plus grand au niveau des PINNs. Ceci est peut-être du aussi à la présence de plus de paramètres. On peut déduire que l'optimisation au niveau du HDES sera d'une complexité moindre qu'au niveau des PINNs.

Conclusion et Perspectives

Dans ce mémoire, nous avons étudié en parallèle deux méthodes de résolution des équations aux dérivées partielles que sont les Physics-Informed Neural Networks (PINNs) et l'Hybrid Differential Equation Solver (H-DES). Notre étude s'est faite sur le cas spécifique de l'équation de la Flamme. Ainsi, nous avons d'abord introduit les réseaux de neurones puis les PINNs et leur fonctionnement avec une application à la résolution d'un système d'équations différentielles. Après nous avons introduit des notions sur l'informatique quantique, l'algorithme variationnel quantique (VQA) et le fonctionnement de l'HDES. Nous appliquons ensuite ces méthodes à l'équation de la flamme tout en effectuant une comparaison sur leurs points communs.

Pour les perspectives, il serait intéressant de poursuivre nos recherches sur les PINNs utilisant les réseaux de neurones convolutionnels (CNN), les réseaux de neurones récurrents (RNN) ou les graphes neuronaux (GNN). Après nous pourrons nous pencher sur la résolution de l'équation de Shrodinger pour des dimensions supérieurs à 1. En effet des méthodes de résolution de cette équation avec les réseaux de neurones convolutionnels et les graphes neuronaux ont vu le jour.

Table des figures

1 La solution QUICK II
2 Flux de travail du QUICK II
3 le marché de ColibriTD III

1.1 Perceptron 4
1.2 Perceptron en forme de neurone 5
1.3 Architecture d'un réseau de neurones artificiels 6

2.1 Définition de l'oracle d'une fonction f 14
2.2 circuit de l'algorithme Deutsch-Jozsa 15
2.3 diagramme du fonctionnement d'un VQA [16] 19

3.1 Fonctionnement des PINNs [23] 23
3.2 Evolution du coût en fonction du nombre d'itérations 25
3.3 Courbes des solutions du système. 25

4.1 Fonctionnement de l'algorithme H-DES 26
4.2 Hardware efficient Ansatz 27
4.3 Solution pour l'équation différentielle linéaire 32

5.1 Courbe de f (Encodage des conditions dans le coût) 35
5.2 Courbe de g (Encodage des conditions dans le coût) 35
5.3 Solution avec une condition stricte. Courbe de f 36
5.4 solution avec une condition stricte. Courbe de g 37
5.5 courbe de f (Fonction d'activation adaptative). 38
5.6 courbe de g (Fonction d'activation adaptative). 38
5.7 distribution des erreurs pour f. 39
5.8 distribution des erreurs pour g. 40
5.9 Distribution des erreurs pour f avec condition en 0.5. 41
5.10 Distribution des erreurs pour g avec condition en 0.5. 41
5.11 Distribution des erreurs pour f avec condition en 0.5. 42

5.12 Distribution des erreurs pour g avec condition en 0.5. 43
5.13 Solution finale f. 44
5.14 Solution finale g. 45
5.15 courbe f avec n = 3, d = 2. 46
5.16 courbe g avec n = 3, d = 2. 47
5.17 courbe f avec n = 3, d = 2 avec 100 itérations. 48
5.18 courbe g avec n = 3, d = 2 avec 100 itérations. 48
5.19 courbe f avec n = 3, d = 5 avec 100 itérations. 49
5.20 courbe g avec n = 3, d = 5 avec 100 itérations. 50
5.21 courbe de f avec H-DES. 51
5.22 courbe de g avec H-DES. 51
5.23 Courbe de f avec 30 paramètres 53
5.24 Courbe de g avec 30 paramètres 53

Bibliographie

[1] Arnaud BODIN. *Quantum un peu de mathématiques pour l'informatique quantique.* Exo7.

[2] D. I. Fotiadis E. Lagaris, A. Likas. Artificial neural networks for solving ordinary and partial differential equations. 1997.

[3] Warren McCullock et Walter Pitts. A logical calculus of the ideas immanent in nervous activity. 1943.

[4] A.Rozza F. Manessi. Learning combinations of activation functions. *in : 2018 24th international conference on pattern recognition (ICPR), IEEE, 2018, pp. 61–66.*, 2018.

[5] Wang JX Gao H, Sun L. Physics-informed geometryadaptive convolutional neural networks for solving parameterized steadystate pdes on irregular domain. *Journal of Computational Physics 428 :110,079.*, 2021.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward. *in Aistats, volume 9, pages 249–256*, 2020.

[7] Andrea Griewank and Andrea Walther. Evaluating derivatives : principles and techniques of algorithmic differentiation. *SIAM*, 2008.

[8] Pavlos Protopapas Henry Jin, Marios Mattheakis. Unsupervised neural networks for quantum eigenvalue problems. 2020.

[9] Frédéric Holweck. Classical and quantum algorithm. Présentations PowerPoint et notes de cours, 2023. Cours dispensé à l'Université UTBM, Pôle Energie et Informatique.

[10] Shiji Song Honghui Wang, Lu Lu and Gao Huang. Learning specialized activation functions for physics-informed neural networks. *Commun. Comput. Phys., 34 (2023), pp. 869-906*, 2023.

[11] https ://learning.quantum.ibm.com/course/variational-algorithm-design/variational algorithms.

[12] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization. *CoRR, abs/1412.6980*, 2014.

[13] Thomas Hanne Kit Wong1, Rolf Dornberger. An analysis of weight initialization methods in connection with different activation functions for feedforward neural networks. 2022.

[14] Siddharth Krishna Kumar. On weight initialization in deep neural networks. 2017.

[15] H. Finger S. Fullhase G. Pipa L. R. Sutfeld, F. Brieger. Adaptive blending units : Trainable activation functions for deep neural networks. *Science and Information Conference, Springer, 2020, pp. 37–50*, 2020.

[16] Ryan Babbush Simon C. Benjamin Suguru Endo Keisuke Fujii Jarrod R. McClean Kosuke Mitarai Xiao Yuan Lukasz Cincio M. Cerezo, Andrew Arrasmith and Patrick J. Coles. Variational quantum algorithms. 2021.

[17] R. Ptucha M. Dushkoff. Adaptive activation functions for deep networks. *Electronic Imaging 2016 (19) (2016) 1–5.*, 2016.

[18] Paris Perdikaris Maziar Raissi and George Em Karniadakis. Physics informed deep learning (part i) : Data-driven solutions of nonlinear partial differential equations. 2017.

[19] Quoc V. Le Prajit Ramachandran, Barret Zoph. Searching activation functions. *Google Brain*, 2017.

[20] Quoc V. Le Prajit Ramachandran, Barret Zoph. Swish : A self-gated activation function. 2017.

[21] F. Rosenblatt. The perceptron, a perceiving and recognizing automation. *Volume 85,Numéros 460 à 461 de Report : Cornell Aeronautical Laboratory*, 1957.

[22] C. Liu S. Wu H. San Wong S. Qian, H. Liu. Adaptive activation functions in convolutional neural networks. *Neurocomputing 272 (2018) 204–212*, 2018.

[23] Fabio Giampaolo Gianluigi Rozza Maziar Raissi Salvatore Cuomo, Vincenzo Schiano Di Cola and Francesco Piccalli. Scientific machine learning through physics-informed neural networks : Where we are and what's next. 2022.

[24] Hanwen Wang Sifan Wang, Shyam Sankaran and Paris Perdikaris. An expert's guide to training physics-informed neural networks. 2023.

[25] Michał Stęchły. Introduction to variational quantum algorithms. 2024.

[26] Dourado A et al Viana FAC, Nascimento RG. Estimating model inadequacy in ordinary differential equations with physicsinformed neural networks. *Computational Materials Science 180 :109,687*, 2020.

[27] Mohamed Al-Sarem Faisal Saeed Moez Krichen Wadii Boulila, Maha Driss. Weight initialization techniques for deep learning algorithms in remote sensing : Recent trends and future perspectives. 2021.

[28] Karniadakis GE Yang L, Zhang D. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing 42(1) :A292-A317*, 2020.