Rapport de fin de stage

Résolution de l'équation de précession d'un moment magnétique par réseaux de neurones

Matthieu Carreau
Telecom Paris, Institut Polytechnique de Paris
F-91120, Palaiseau, France
matthieu.carreau@telecom-paris.fr

supervisé par
Stam Nicolis
Institut Denis Poisson
Université de Tours, Université d'Orléans, CNRS (UMR7013)
Parc de Grandmont, F-37200, Tours, France

Pascal Thibaudeau
CEA Le Ripault
BP 16, F-37260, Monts, France

Juillet 2022

Résumé

Ce rapport présente une méthode numérique permettant l'étude de la précession d'un moment magnétique dans un champ extérieur en présence d'un amortissement. On décrira et mettra en œuvre des méthodes d'optimisation pour déterminer des solutions approchées d'équations différentielles, éventuellement non-linéaires, en fixant un modèle de solutions impliquant un réseau de neurones. En particulier, on montrera une méthode de descente de gradients qui permet de minimiser une fonction de coût servant à approcher la solution de l'équation différentielle. La mise en œuvre numérique sera ensuite comparée à une suite de problèmes formée par ordre de difficulté croissante. D'une part nous chercherons à retrouver une primitive d'une fonction trigonométrique, puis nous évaluerons la solution d'un système de deux équations différentielles couplées, décrivant la précession de Larmor. Enfin nous évaluerons notre approche sur l'équation de Landau-Lifshitz-Gilbert, qui décrit le phénomène d'amortissement.

Table des matières

Chapitre 1

Introduction

Ce rapport est le compte-rendu d'un stage d'un mois à l'Institut Denis Poisson de Tours ayant pour objet la mise en place et l'implémentation de méthodes de résolution d'équations différentielles à l'aide de réseaux de neurones. Le problème physique étudié est celui de la précession d'un moment magnétique dans un champ magnétique, que l'on décrira dans un premier temps par l'équation de précession de Larmor [1], puis par l'équation de Landau-Lifshitz-Gilbert [2]. Cette seconde version de l'équation possède une solution analytique obtenue par projection transverse dans le cas d'une precession dans un champ statique et homogène mais on ne lui connaît pas de solution analytique en dehors de ce cas. C'est pourquoi il est pertinent de se pencher sur des méthodes numériques pour en obtenir des solutions approchées.

L'approche proposée ici consiste à utiliser des réseaux de neurones, connus pour leur capacité à approcher des fonctions [6], pour trouver des fonctions "proches" d'être solution d'une équation différentielle. On utilise par conséquent les réseaux de neurones en suivant les références [3, 4, 5], en utilisant les algorithmes d'optimisation connus pour minimiser une fonction de coût (désignée aussi sous la nom de fonction erreur), qui indique dans quelle mesure une fonction candidate, construite à partir du réseau de neurones, est proche de la solution.

On écrira lorsque cela sera possible l'équation différentielle d'inconnue $\boldsymbol{M} : \mathbb{R} \to \mathbb{R}^n$ (vecteur adimensionné représentant le moment magnétique) sous une forme similaire à la suivante, où $\boldsymbol{g} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ est une fonction connue :

$$\begin{cases} \dfrac{d\boldsymbol{M}(t)}{dt} = \boldsymbol{g}(t, \boldsymbol{M}(t)) \\ \boldsymbol{M}(0) = \boldsymbol{M}_0 \end{cases} \tag{1.1}$$

La méthode consiste ensuite à construire une fonction candidate pour l'équation différentielle comme une somme de deux termes. Le premier est un terme non ajustable vérifiant la condition initiale, et le deuxième dépendant des paramètres à entraîner représentés par le vecteur $\boldsymbol{P}$. Ce dernier est construit de façon à ne pas influencer la valeur initiale de la fonction, afin que la condition initiale soit toujours satisfaite, ceci étant réalisé grâce à une fonction $f : \mathbb{R} \to \mathbb{R}$ fixée à l'avance (souvent $f = id_{\mathbb{R}}$)

$$\tilde{\boldsymbol{M}}(t) = \boldsymbol{M}_0 + f(t)\mathcal{N}(t, \boldsymbol{P}) \tag{1.2}$$

On définit ensuite une fonction de coût, que l'on calcule à partir de N points $(t_i)_{i\in[\![1,N]\!]}$ de l'intervalle d'étude, de la forme :

$$E(\boldsymbol{P}) = \sum_{i=1}^{N} (\frac{d\tilde{\boldsymbol{M}}}{dt}(t_i) - g(t_i, \tilde{\boldsymbol{M}}(t_i)))^2 \tag{1.3}$$

Il s'agira enfin d'utiliser différentes méthodes d'optimisation propres aux réseaux de neurones afin de trouver un vecteur $\boldsymbol{P}$ qui minimise (et idéalement annule) cette fonction d'erreur.

Dans le chapitre 2, on s'intéressera à résoudre une équation différentielle à une dimension, en choisissant d'abord le terme ajustable $\mathcal{N}(t, \boldsymbol{P})$ sous la forme d'une série de Fourier, puis comme la sortie d'un réseau de neurones. On appliquera dans le chapitre 3 les mêmes méthodes pour la résolution d'un système de deux équations différentielles couplées représentant le mouvement de précession décrit par l'équation de Larmor. On étudiera ensuite dans le chapitre 4 l'effet de l'ajout du terme d'amortissement introduit dans l'équation de Landau-Lifshitz-Gilbert. Enfin dans le chapitre 5, je discuterai des résultats obtenus et proposerai quelques perspectives.

Chapitre 2

Equation différentielle à une dimension

On souhaite se familiariser avec différentes méthodes numériques appliquées à la résolution d'une équation différentielle d'ordre 1 en une dimension.

On considère alors M, une fonction réelle à une variable satisfaisant l'équation différentielle suivante, où M_0 désigne la valeur initiale de la fonction :

$$\begin{cases} \dfrac{dM(t)}{dt} + \cos(2\pi t) = 0 \\ M(0) = M_0 \end{cases} \tag{2.1}$$

Cherchons à mettre en œuvre les méthodes présentées dans la section 1 sur l'équation (2.1), pour tout $t \in [t_a = 0, t_b = 1]$. L'équation (2.1) et sa condition initiale donnée en $t = 0$, admettent une solution analytique unique qui s'écrit

$$M(t) = M_0 - \frac{1}{2\pi}\sin(2\pi t) \tag{2.2}$$

L'équation (2.2) permettra d'évaluer la qualité des solutions numériques trouvées par différence.

2.1 Solution en séries de Fourier

On cherche des solutions numériques approchées de l'équation (2.1) sous la forme de séries de Fourier tronquées en un nombre entier H d'harmoniques. On écrit pour cela la solution approchée $\tilde{M}$ comme la somme de deux termes :

— le premier terme est la constante M_0, paramètre non ajustable vérifiant la condition initiale,
— le deuxième terme est $\mathcal{N}(t, \boldsymbol{A})$, une fonction temporelle développée sur une base et dépendant d'un vecteur $\boldsymbol{A}$ de dimension H, dont les valeurs sont notées $(A_m)_{m\in[\![1,H]\!]}$. Cette fonction est construite de façon à ne pas influencer la condition initiale de la fonction.

Ainsi on s'assure en particulier que $\mathcal{N}(0, \boldsymbol{A}) = 0$ de sorte que la solution approchée coïncide avec la solution exacte à $t = 0$. En équations cela donne

$$\begin{cases} \tilde{M}(t) = M_0 + \mathcal{N}(t, \boldsymbol{A}) \\ \mathcal{N}(t, \boldsymbol{A}) = \sum_{m=1}^{H} A_m \sin(2\pi m t) \end{cases} \tag{2.3}$$

On définit une fonction de coût E à minimiser dans un espace L^2, obtenue à partir de la valeur de la dérivée de $\tilde{M}$ sur un intervalle temporel de N points pour t. Nous posons $\forall i \in [\![1, N]\!], t_i = \frac{i-1}{N}$:

$$\begin{aligned} E &= \frac{1}{2}\sum_{i=1}^{N}(\frac{d\tilde{M}}{dt}(t_i) + \cos(2\pi t_i))^2 \\ &= \frac{1}{2}\sum_{i=1}^{N}(\sum_{m=1}^{H} 2\pi m A_m \cos(2\pi m t_i) + \cos(2\pi t_i))^2 \end{aligned} \tag{2.4}$$

Puisque le vecteur $\boldsymbol{A}$ représente maintenant un ensemble de paramètres ajustables, on cherche le minimum de E en tant que fonction de $\boldsymbol{A}$. Une condition nécessaire sur $\boldsymbol{A}$ pour être un antécédent d'un minimum est

$$\frac{\partial E}{\partial A_l} = 0, \forall l \in [\![1, H]\!] \tag{2.5}$$

On doit maintenant déterminer les dérivées partielles de E par rapport à chacun des coefficients de $\boldsymbol{A}$. Ces dérivées partielles sont données pour $l \in [\![1, H]\!]$ par :

$$\frac{\partial E}{\partial A_l} = \sum_{i=1}^{N}(\sum_{m=1}^{H} 2\pi m A_m \cos(2\pi m t_i) + \cos(2\pi t_i))2\pi l \cos(2\pi l t_i) \tag{2.6}$$

Nous devons donc trouver les coefficients A_m avec $m \in [\![1, H]\!]$ qui vérifient la condition 2.5, et vérifier que le vecteur des coefficients trouvés correspond bien à la solution analytique, i.e $\forall m \in [\![1, H]\!], A_m = -\frac{1}{2\pi}\delta_1^m$. Pour cela nous allons développer deux approches. Dans la section 2.1.1 nous inverserons un système linéaire et dans la section 2.1.2 nous formerons un algorithme fondé sur une descente de gradients.

2.1.1 Inversion d'un système linéaire

Pour la fonction approchée $\tilde{M}$, la condition 2.5 qui est équivalente à l'expression 2.6, est en réalité un système linéaire de H équations, que l'on peut représenter en définissant les matrices suivantes :

$$\mathcal{M} = (r_{m,l})_{(m,l)\in[\![1,H]\!]^2}, \boldsymbol{A} = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_H \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_H \end{pmatrix} \tag{2.7}$$

$$\forall (m,l) \in [\![1,N]\!]^2, \begin{cases} r_{m,l} = 2\pi ml \displaystyle\sum_{i=1}^{N} \cos(2\pi m t_i)\cos(2\pi l t_i) \\ b_l = -l \displaystyle\sum_{i=1}^{N} \cos(2\pi t_i)\cos(2\pi l t_i) \end{cases} \tag{2.8}$$

On résoud ce système linéaire en écrivant l'équation matricielle qui le représente, c'est-à-dire :

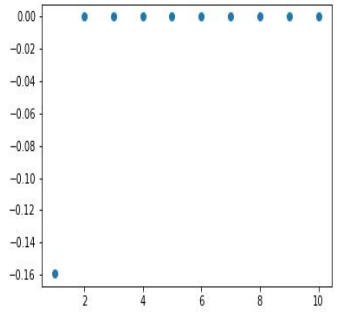
$$\mathcal{M}\boldsymbol{A} = \boldsymbol{b} \Leftrightarrow \boldsymbol{A} = \mathcal{M}^{-1}\boldsymbol{b} \tag{2.9}$$

Il ne reste plus maintenant qu'à réaliser l'implémentation numérique associée en python, en instanciant la matrice $\mathcal{M}$ et les vecteurs $\boldsymbol{A}$ et $\boldsymbol{b}$ à l'aide de la bibliothèque *numpy*.

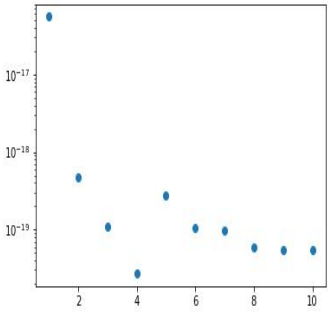
Malheureusement la matrice $\mathcal{M}$ n'est pas toujours inversible selon le choix de H et N. En effet, on peut montrer que c'est nécessairement le cas lorsque $H > N$. Il suffit pour cela d'écrire la matrice $\mathcal{M}$ en fonction de la matrice $\mathcal{A}$ de taille $N \times H$, dont les coefficients sont données par $\forall (i,j) \in [\![1,N]\!] \times [\![1,H]\!], a_{i,j} = j\cos(2\pi j t_i)$. On a alors $\mathcal{M} = 2\pi(\mathcal{A}^T)\mathcal{A}$. On déduit un majorant du rang de $\mathcal{M}$: $\mathrm{rg}(\mathcal{M}) \leq \min(\mathrm{rg}(\mathcal{A}^T), \mathrm{rg}(\mathcal{A})) = \mathrm{rg}(\mathcal{A}) \leq N$. Ainsi si $N < H$, $\mathcal{M}$ n'est pas inversible.

De plus, cette matrice n'est pas non plus inversible lorsque H et N sont proches l'un de l'autre, par exemple pour $H = N = 10$. Il serait intéressant d'approfondir cet aspect pour savoir si cela se traduit par le fait que la fonction E admette plusieurs minima locaux.

En réalisant des expériences numériques, il semble que choisir $N \gg H$ soit suffisant pour que $\mathcal{M}$ soit inversible. On choisira $H = 10, N = 100$ pour la suite. On obtient alors les coefficients présentés dans la figure 2.1.b avec les valeurs absolues des erreurs de chacun des coefficients trouvés par rapport à leur valeur exacte (Figure 2.1.b). On constate que les erreurs sur chaque



(a) Valeur des coefficients A_l



(b) Erreur des coefficients A_l

FIGURE 2.1 – Détermination des 10 coefficients du vecteur $\boldsymbol{A}$ sur une base temporelle donnée, par inversion d'un système linéaire.

coefficient est inférieure à 10^{-16}, ce qui représente la précision machine. On valide donc cette première méthode.

2.1.2 Algorithme de descente de gradients

Profitons de cet exemple simple où le système est linéaire pour se familiariser avec l'algorithme de descente de gradients qui sera utile par la suite. Le principe est de partir d'un vecteur $\boldsymbol{A}^{(0)}$ choisit aléatoirement puis de calculer itérativement un nouveau vecteur $\boldsymbol{A}^{(k)}$ en se déplaçant dans l'espace dans la direction qui minimise le plus l'erreur localement. On définit pour cela les paramètres suivants :

$$\alpha > 0, \boldsymbol{A}^{(0)} = \begin{pmatrix} A_1^{(0)} \\ A_2^{(0)} \\ \vdots \\ A_H^{(0)} \end{pmatrix}, \boldsymbol{g}^{(0)} = \begin{pmatrix} \frac{\partial E^{(0)}}{\partial A_1^{(0)}} \\ \frac{\partial E^{(0)}}{\partial A_2^{(0)}} \\ \vdots \\ \frac{\partial E^{(0)}}{\partial A_H^{(0)}} \end{pmatrix}, \tag{2.10}$$

Puis on calcule itérativement :

$$\boldsymbol{A}^{(k+1)} = \boldsymbol{A}^{(k)} - \alpha \boldsymbol{g}^{(k)} \tag{2.11}$$

On cherche à trouver le coefficient α optimal qui assure la convergence tout en maximisant la vitesse de convergence. On exprime tout d'abord le gradient en fonction de la matrice $\mathcal{M}$ et du vecteur $\boldsymbol{b}$ définis précédemment qui sont indépendants de $\boldsymbol{A}$ et de k :

$$\boldsymbol{g}^{(k)} = \mathcal{M}\boldsymbol{A}^{(k)} - \boldsymbol{b} \tag{2.12}$$

Ainsi, l'équation de récurrence (2.11) se réécrit comme une suite arithmético-géométrique de vecteurs :

$$\boldsymbol{A}^{(k+1)} = (\mathcal{I}_H - \alpha\mathcal{M})\boldsymbol{A}^{(k)} + \alpha\boldsymbol{b} \tag{2.13}$$

Cette équation de récurrence peut se réécrire si $\mathcal{M}$ est inversible en posant $\boldsymbol{A^*} = \mathcal{M}^{-1}\boldsymbol{b}$ (qui est l'unique solution du système linéaire) on arrive à l'expression de $\boldsymbol{A}^{(k)} - \boldsymbol{A^*}$ comme une suite géométrique :

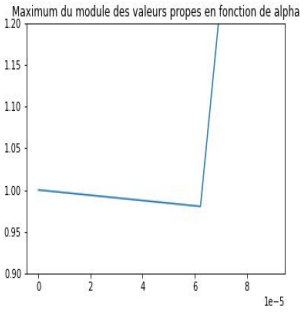
$$\begin{aligned} \boldsymbol{A}^{(k+1)} - \boldsymbol{A^*} &= (\mathcal{I}_H - \alpha\mathcal{M})\boldsymbol{A}^{(k)} + \alpha\boldsymbol{b} - \mathcal{M}^{-1}\boldsymbol{b} \\ &= (\mathcal{I}_H - \alpha\mathcal{M})(\boldsymbol{A}^{(k)} - \boldsymbol{A^*}) \\ &= (\mathcal{I}_H - \alpha\mathcal{M})^k(\boldsymbol{A}^{(0)} - \boldsymbol{A^*}) \end{aligned} \tag{2.14}$$

Pour s'assurer de la convergence de cette suite, on s'intéresse au rayon spectral, c'est-à-dire le maximum du module des valeurs propres, de $\mathcal{R}_\alpha = (\mathcal{I}_H - \alpha\mathcal{M})$, noté $\rho(\mathcal{R}_\alpha)$. La suite $(\boldsymbol{A}^{(k)})_{k\in\mathbb{N}}$ converge si et seulement si la norme de $(\mathcal{R}_\alpha^k)_{k\in\mathbb{N}}$ tend vers 0, sa limite est alors $\boldsymbol{A}^*$. Cela est le cas si et seulement si $\rho(\mathcal{R}_\alpha) < 1$ [8]. De plus, elle convergera d'autant plus vite que son rayon spectral est faible. On trace donc ce maximum en fonction de α en figure 2.2. On en déduit la valeur critique $\alpha_c = 6.2807.10^{-5}$, pour laquelle $\rho(\mathcal{R}_\alpha) = 1$, ainsi que la valeur $\alpha_{min} = 6.2189.10^{-5}$ pour $\rho(\mathcal{R}_\alpha)$ est minimal.

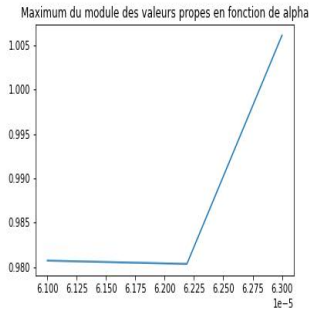
On éxécute l'algorithme en parallèle pour les deux valeurs de α trouvées précédemment ainsi que pour une valeur $\alpha_1 = 6.3.10^{-5}$, tel que $\rho(\mathcal{R}_\alpha) > 1$ On montre l'évolution de l'erreur en fonction du nombre d'itérations en figure 2.3. On constate que α_{min} donne lieu à une décroissance exponentielle de l'erreur pendant les 2000 premières itérations, qui devient ensuite stationnaire. α_1 donne une erreur qui croît exponentiellement car la norme de $(\mathcal{R}_\alpha^n)_{n\in\mathbb{N}}$ diverge exponentiellement. La valeur α_c donne une erreur constante, et correspond au cas limite entre les 2 cas précédents.

On retient donc les résultats obtenus pour α_{min} que l'on montre en figure 2.4 avec les valeurs absolues des erreurs de chacun par rapport à la valeur théorique. On constate que les erreurs sur chaque coefficient est inférieure à 10^{-16}, soit la précision machine et on valide donc cette seconde méthode.

Cette étude a mis en évidence l'importance du choix du taux d'apprentissage qui détermine le comportement de la suite de vecteurs obtenus lors de la descente de gradients. Cela sera important à garder en tête par la suite, même quand les systèmes à résoudre ne seront plus linéaire et qu'il ne sera plus possible de déterminer à l'avance les valeurs optimales. Elles devront donc être testées empiriquement.



(a) $\alpha \in [0, 10^{-4}]$



(b) Au voisinage de l'intersection avec 1

FIGURE 2.2 – Rayon spectral de $\mathcal{R}_\alpha$ en fonction de α

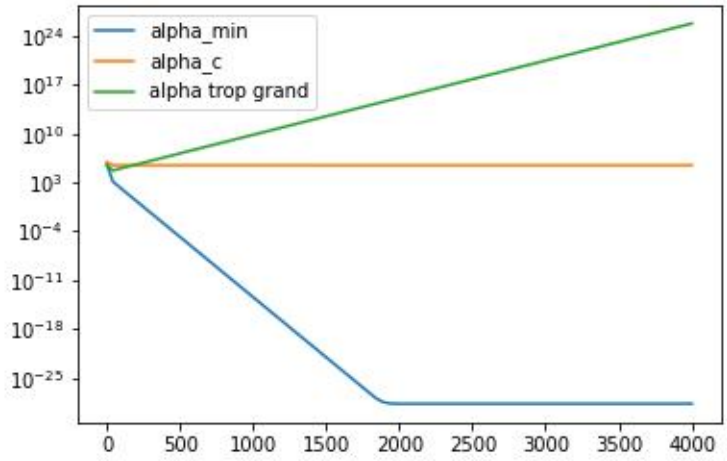
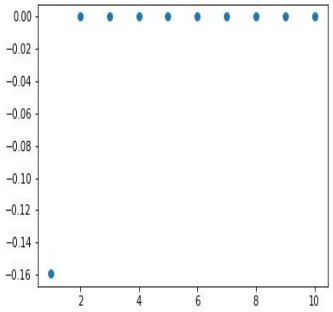
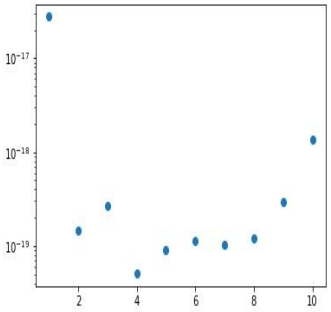


FIGURE 2.3 – Erreurs en fonction du nombre d'itérations pour 3 valeurs de α



(a) Coefficients trouvés



(b) Valeurs absolue de l'erreur pour chaque coefficient

FIGURE 2.4 – Résultats de la méthode de descnte de gradients

2.2 Solutions par réseau de neurones

On cherche à présent à utiliser un réseau de neurones pour approcher la solution de l'équation différentielle. On cherche désormais des solutions approchées sous la forme suivante :

$$\begin{cases} \tilde{M}(t) = M_0 + t\mathcal{N}(t, \boldsymbol{P}) \\ \mathcal{N}(t, \boldsymbol{P}) = \sum_{j=1}^{H} v_j \sigma(w_j t + b_j) \end{cases} \tag{2.15}$$

Le terme $\mathcal{N}(t, \boldsymbol{P})$ correspond donc à la sortie d'un réseau de neurones dont l'architecture est présentée en figure 2.5, contenant une couche cachée intermédiaire, qui réalise en sortie une somme pondérée de sigmoïdes, la fonction utilisée est $\forall x \in \mathbf{R}, \sigma(x) = \dfrac{1}{1+e^{-x}}$. Les paramètres $\boldsymbol{P}$ à ajuster sont désormais les coefficients $(w_j)_{j\in[\![1,H]\!]}$, $(b_j)_{j\in[\![1,H]\!]}$ et $(v_j)_{j\in[\![1,H]\!]}$.

Pour plus de détails, la page [7] présente le principe de ce type de réseau de neurones.

On définit une nouvelle fonction d'erreur, calculée à partir des N points suivants : $\forall i \in [\![1, N]\!], t_i = \frac{i-1}{N-1}$

$$E(\boldsymbol{P}) = \sum_{i=1}^{N} \left(\frac{d\tilde{M}}{dt}(t_i) + \cos(2\pi t_i)\right)^2 \tag{2.16}$$

On calcule ensuite les expressions analytiques des dérivées partielles de $E(\boldsymbol{P})$ par rapport à chaque paramètre ajustable, puis on cherche à minimiser cette erreur à l'aide de l'algorithme de descente de gradients.

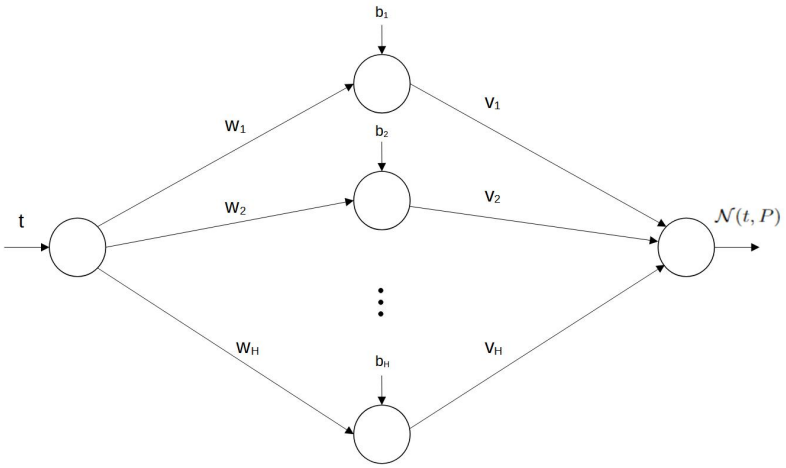


FIGURE 2.5 – Réseau de neurones

2.2.1 Résultats obtenus

On initialise l'algorithme avec les paramètres suivants : $(H = 4, N = 20)$ On obtient une erreur de $1,2.10^{-2}$ et une estimation visible en figure 2.6. Cela permet de valider notre modèle sur l'étude à une dimension.

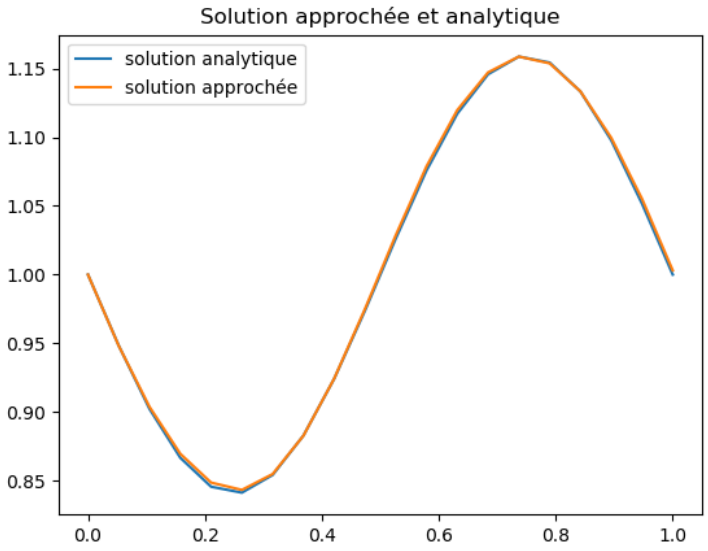


FIGURE 2.6 – estimation de la solution par un réseau de neurones

Chapitre 3

Mouvement de précession

On s'intéresse désormais à l'équation de la précession de Larmor [1] pour un moment magnétique $\boldsymbol{M} \in \mathbb{R}^3$ autour d'un vecteur $\boldsymbol{\omega}$ pointant le long de l'axe z et dont la composante sur cet axe est constante, soit $\boldsymbol{\omega} = \omega_z \boldsymbol{z}$.

$$\frac{d\boldsymbol{M}}{dt} = \boldsymbol{\omega} \times \boldsymbol{M} \tag{3.1}$$

Cette équation vectorielle implique que $\frac{d\boldsymbol{M}}{dt}$ et $\boldsymbol{\omega}$ sont deux vecteurs orthogonaux, ainsi $\frac{dM_z}{dt} = 0$. Par conséquent, on ne considérera que les composantes selon les vecteurs $\boldsymbol{x}$ et $\boldsymbol{y}$ et l'équation 3.1 se réécrit alors comme un système de deux équations différentielles couplées :

$$\begin{cases} \dfrac{dM_x}{dt} = \omega M_y \\ \dfrac{dM_y}{dt} = -\omega M_x \end{cases} \tag{3.2}$$

On cherchera les solutions pour $t \in [t_a = 0, t_b = 1]$, en imposant les conditions initiales suivantes :

$$\begin{cases} M_x(0) = M_0 \\ M_y(0) = 0 \end{cases} \tag{3.3}$$

On pourra ensuite comparer nos résultats avec la solution analytique connue de ce système qui vaut :

$$\begin{cases} M_x(t) = M_0 \cos(2\omega t) \\ M_y(t) = -M_0 \sin(2\omega t) \end{cases} \tag{3.4}$$

3.1 Solutions en séries de Fourier

On cherche des solutions numériques approchées sous la forme de séries de Fourier tronquées avec H harmoniques, en posant la forme suivante :

$$\begin{cases} \tilde{M}_x(t) = M_0 + \sum_{m=1}^{H} A_m(\cos(m\omega t) - 1) + B_m \sin(m\omega t) \\ \tilde{M}_y(t) = \sum_{m=1}^{H} -A_m \sin(m\omega t) + B_m(\cos(m\omega t) - 1) \end{cases} \tag{3.5}$$

Les coefficients $(A_m)_{m\in[\![1,H]\!]}$ et $(B_m)_{m\in[\![1,H]\!]}$ des vecteurs $\boldsymbol{A}$ et $\boldsymbol{B}$ sont les paramètres à ajuster. On cherche à obtenir la solution analytique, i.e $\forall m \in [\![1, H]\!], A_m = \delta_1^m$ et $\forall m \in [\![1, H]\!], B_m = 0$.

La fonction de coût pour ces solutions potentielles, en s'intéressant aux N points suivants : $\forall i \in [\![1, N]\!], t_i = \frac{i-1}{N}$ s'écrit

$$E(\boldsymbol{P}) = \sum_{i=1}^{N} (\frac{d\tilde{M}_x}{dt}(t_i) - \omega \tilde{M}_y(t_i))^2 + (\frac{d\tilde{M}_y}{dt}(t_i) + \omega \tilde{M}_x(t_i))^2, \tag{3.6}$$

où $\boldsymbol{P} = \boldsymbol{A} \cup \boldsymbol{B}$.

En utilisant la méthode de descente de gradients, nous calculons les dérivées partielles suivantes : $(\frac{\partial E}{\partial A_l}, \frac{\partial E}{\partial B_l})_{l\in[\![1,H]\!]}$

On peut se ramener à l'écriture du cas à une dimension définissant le vecteur $\boldsymbol{P}$ comme l'union des vecteurs $\boldsymbol{A}$ et $\boldsymbol{B}$ et le vecteur $\boldsymbol{g}$ comme le vecteur contenant les dérivées partielles de E par rapport à chaque composante de $\boldsymbol{P}$.

$$\boldsymbol{P}^{(k)} = \begin{pmatrix} A_1^{(k)} \\ \vdots \\ A_H^{(k)} \\ B_1^{(k)} \\ \vdots \\ B_H^{(k)} \end{pmatrix}, \boldsymbol{g}^{(k)} = \begin{pmatrix} \frac{\partial E^{(k)}}{\partial A_1^{(k)}} \\ \vdots \\ \frac{\partial E^{(k)}}{\partial A_H^{(k)}} \\ \frac{\partial E^{(k)}}{\partial B_1^{(k)}} \\ \vdots \\ \frac{\partial E^{(k)}}{\partial B_H^{(k)}} \end{pmatrix}, \tag{3.7}$$

Il existe alors une nouvelle matrice $\mathcal{M}$ d'ordre $2H$ ainsi qu'un nouveau vecteur $\boldsymbol{b}$ de taille $2H$ tels que les équations définissant la descente de gradients soient les suivantes :

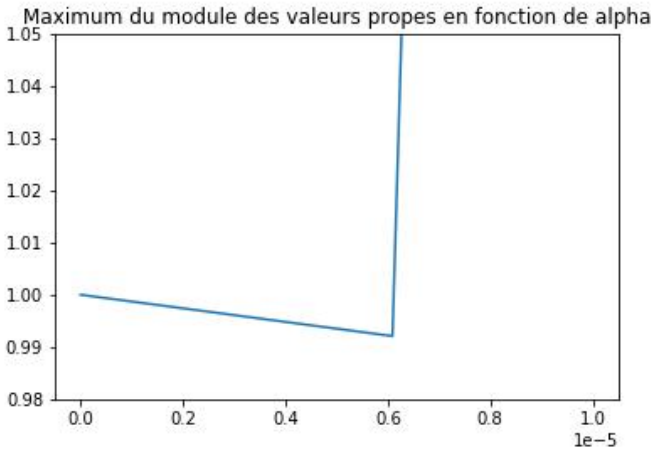


FIGURE 3.1 – Evolution du maximum du module en fonction de α

$$\boldsymbol{P}^{(k+1)} = \boldsymbol{P}^{(k)} - \alpha \boldsymbol{g}^{(k)} \tag{3.8}$$

$$\boldsymbol{g}^{(k)} = \mathcal{M}\boldsymbol{P}^{(k)} - \boldsymbol{b} \tag{3.9}$$

$$\boldsymbol{P}^{(k+1)} = (\mathcal{I}_H - \alpha\mathcal{M})\boldsymbol{P}^{(k)} + \alpha\boldsymbol{b} \tag{3.10}$$

On réalise la même étude qu'en dimension 1 pour la recherche du coefficient α_{min} qui minimise le maximum des modules des valeurs propres de $\mathcal{R}_\alpha = \mathcal{I}_H - \alpha\mathcal{M}$, ainsi que le coefficient α_c à ne pas dépasser, à partir duquel ce maximum et supérieur à 1 et la suite diverge. On trouve les valeurs suivantes $\alpha_{min} = 6.0906.10^{-6}$ et $\alpha_c = 6.1146.10^{-6}$, et l'évolution du maximum en fonction de α est montré en figure 3.1.

On éxécute l'algorithme en choisissant la valeur α_{min} précédente et on montre l'évolution de l'erreur en fonction du nombre d'itérations en figure 3.2. On constate comme dans le cas à 1 dimension que α_{min} donne lieu à une décroissance exponentielle de l'erreur, cette fois pendant les 4500 premières itérations, qui devient ensuite quasiment stationnaire, de l'ordre de 10^{-25}.

Les coefficients finaux de $\boldsymbol{P}$ ainsi que les valeurs absolues de leurs erreurs sont représentées en figure 3.3. On peut constater que l'on retrouve bien les coefficients attendus, avec une erreur au maximum de l'ordre de 10^{-15}.

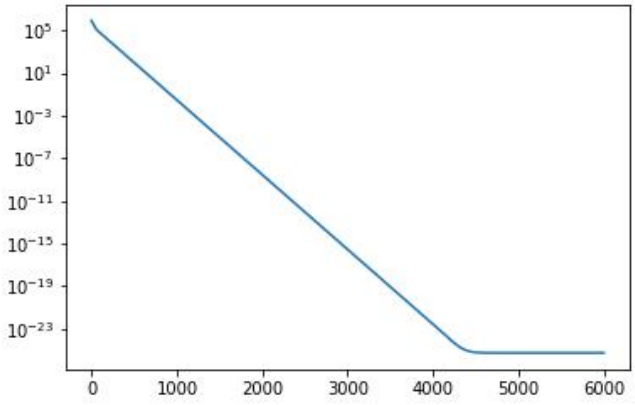
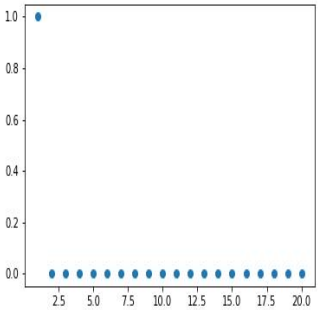
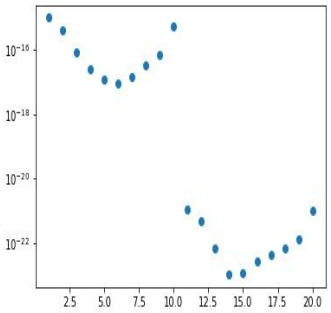


FIGURE 3.2 – Valeur de la fonction d'erreur en fonction du nombre d'itérations



(a) Coefficients trouvés



(b) Valeurs absolue de l'erreur pour chaque coefficient

FIGURE 3.3 – Résultats de la méthode de descente de gradients pour le mouvement de précession

3.2 Solutions aprochées par un réseau de neurones

On cherche dans ce paragraphe à approcher la solution du système des 2 équations grâce à un réseau de neurones. On choisit l'architecture suivante pour le réseau de neurones : 1 entrée pour la variable temporelle, une première couche intermédiaire de $h_1 = 32$ neurones, une seconde couche intermédiaire de $h_2 = 8$ neurones, toutes deux utilisant la sigmoïde définie précédement comme fonction d'activation, et enfin 2 neurones de sortie connectées à la deuxième couche intermédiaire, avec l'identité comme fonction d'activation en sortie. On note les sorties de ces deux derniers neurones respectivement $\mathcal{N}_x(t, \boldsymbol{P})$ et $\mathcal{N}_y(t, \boldsymbol{P})$, où $\boldsymbol{P}$ représente l'ensemble des paramètres du réseau de neurones. On construit alors nos solutions approchées de la façon suivante :

$$\begin{cases} \tilde{M}_x(t) = M_0 + t\mathcal{N}_x(t, \boldsymbol{P}) \\ \tilde{M}_y(t) = t\mathcal{N}_y(t, \boldsymbol{P}) \end{cases} \tag{3.11}$$

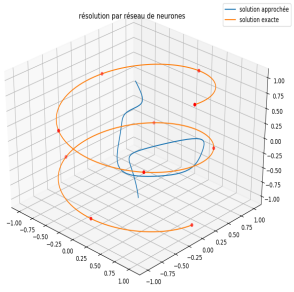
On définit une fonction d'erreur pour ces solutions potentielles, en s'intérressant aux N points suivants de l'intervalle $[t_a = -1, t_b = 1]$: $\forall i \in [\![1, N]\!], t_i = -1 + 2\frac{i-1}{N}$:

$$E(\boldsymbol{P}) = \frac{1}{N}\sum_{i=1}^{N}(\frac{d\tilde{M}_x}{dt}(t_i) - \omega\tilde{M}_y(t_i))^2 + (\frac{d\tilde{M}_y}{dt}(t_i) + \omega\tilde{M}_x(t_i))^2 \tag{3.12}$$

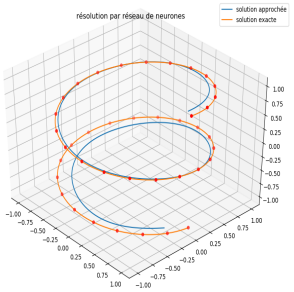
On réalisera à présent les implémentations en python à l'aide de la bibliothèque TensorFlow.

On choisit ensuite les paramètres d'entraînement suivants : taux d'apprentissage : 0.007, 50000 itérations, et on lance 3 entraînements pour $N \in \{10, 40, 100\}$. On obtient les résultats montrés en figure 3.4, où l'axe vertical représente le temps, et les axes horizontaux représentent M_x et M_y. La solution analytique est représentée en orange, la solution approchée est représentée en bleu, et les points rouges sont les points correspondant aux points d'entraînement $(t_i)_{i\in[\![1,N]\!]}$.

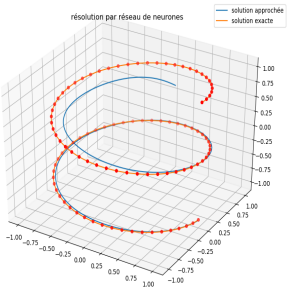
On constate que pour $N = 10$ la solution obtenue est trop éloignée de la solution attendue. Cependant, pour cette durée d'entraînement, il ne semble pas apparaître de différence significative entre $N = 40$ et $N = 100$.



(a) Solution obtenue pour $N = 10$



(b) Solution obtenue pour $N = 40$



(c) Solution obtenue pour $N = 100$

FIGURE 3.4 – Représentations graphiques de M_x et M_y en fonction du temps par le réseau de neurones

Chapitre 4

Précession avec amortissement

Dans ce chapitre, on s'intéresse à déterminer le mouvement de précession d'un moment magnétique $\boldsymbol{M} \in \mathbb{R}^3$ amorti. Son équation d'évolution est représentée en fonction d'une constante d'amortissement réelle et sans dimension notée λ et s'établit, pour simplifier, autour d'un vecteur $\boldsymbol{\omega}$ pointant le long de l'axe z et dont la composante sur cet axe est constante, soit $\boldsymbol{\omega} = \omega_z \boldsymbol{z}$.

L'équation considérée prend le nom d'équation de précession avec un terme d'amortissement introduit par Gilbert [2] et s'écrit

$$\frac{d\boldsymbol{M}}{dt} = (\boldsymbol{\omega} - \lambda \frac{d\boldsymbol{M}}{dt}) \times \boldsymbol{M} \tag{4.1}$$

Cette équation vectorielle admet plusieures propriétés remarquables que nous allons exploiter.

4.1 Projection sur un plan normal

Afin de résoudre l'équation 4.1, une méthode est de prendre le produit vectoriel par $\boldsymbol{M}$ à gauche des deux membres de l'équation 4.1, et de former une projection sur un plan normal à $\boldsymbol{M}$:

$$\boldsymbol{M} \times \frac{d\boldsymbol{M}}{dt} = \boldsymbol{M} \times ((\boldsymbol{\omega} - \lambda \frac{d\boldsymbol{M}}{dt}) \times \boldsymbol{M}) \tag{4.2}$$

Aprés développement, et le changement d'échelle temporelle $\tilde{t} = (1+\lambda^2)t$, on obtient 3 nouvelles équations différentielles scalaires :

$$\begin{cases} \dfrac{dM_x}{d\tilde{t}} = -M_y\omega_z - \lambda M_x M_z \omega_z \\ \dfrac{dM_y}{d\tilde{t}} = M_x\omega_z - \lambda M_y M_z \omega_z \\ \dfrac{dM_z}{d\tilde{t}} = \lambda\omega_z(1 - M_z^2) \end{cases} \tag{4.3}$$

On remarque que l'équation sur M_z est découplée des 2 autres, on cherche ainsi à résoudre celle-ci en premier lieu en section 4.1.1. Puis nous résoudrons les deux équations couplées sur M_x et M_y en section 4.1.2, en y injectant la solution approchée trouvée pour M_z.

4.1.1 Résolution de l'équation sur M_z

On utilise la même approche que précédemment pour résoudre l'équation à l'aide d'un réseau de neurones. On choisit comme architecture du réseau de neurones une entrée pour la variable temporelle, suivie de couches intermédiaires contenant respectivement h_1 et h_2 neurones, connectés à la sortie unique du réseau de neurones, dont les paramètres sont représentés par le vecteur $\boldsymbol{P}$. Comme précédemment, afin de satisfaire la condition initiale, on cherche les fonction $\tilde{M}_z$ sous la forme suivante, où $\mathcal{N}(\tilde{t}, \boldsymbol{P})$ représente la sortie du réseau de neurones.

$$\tilde{M}_z(\tilde{t}) = M_z(0) + \tilde{t}\mathcal{N}(\tilde{t}, \boldsymbol{P}) \tag{4.4}$$

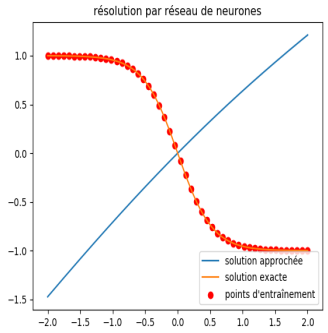
On définit la fonction d'erreur suivante :

$$E(\boldsymbol{P}) = \frac{1}{N}\sum_{i=1}^{N}(\frac{d\tilde{M}_z}{dt}(t_i) - \lambda\omega_z(1 - M_z^2))^2 \tag{4.5}$$

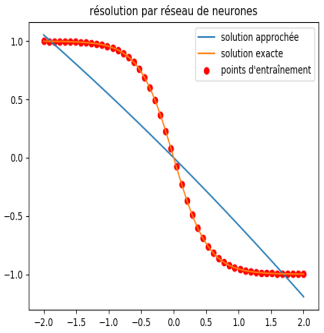
On fixe les paramètres suivants : $\lambda = -0.3$, nombre de points de tests : $N = 30$, $h_1 = 16$, $h_2 = 8$, intervalle d'étude : $[t_a = -1, t_b = 1]$, $M_z(0) = 0$. Lors des essais d'entraînement du modèle on fait face à plusieurs problèmes. On remarque que souvent la sortie du réseau de neurones est quasiment constante sur l'intervalle d'étude. Ainsi, il semble que l'entraînement ne consiste alors dans ce cas qu'à ajuster cette valeur constante pour minimser la fonction d'erreur dans le cas où M_z est linéaire par rapport à $\tilde{t}$, (d'après l'équation 4.4,pour $\mathcal{N}(\tilde{t}, \boldsymbol{P}) = \mu \in \mathbf{R}$ constante, et $M_z(0) = 0$).

Après plusieurs essais d'entraînement, on remarque que l'on tombe sur l'une ou l'autre des 2 solutions présentées en figure 4.1.

On peut vérifier cette hypothèse en exprimant analytiquement notre fonction d'erreur pour une fonction M_z linéaire : $\forall \tilde{t} \in \mathbf{R}, \tilde{M}_z(\tilde{t}) = \mu\tilde{t}$. On peut



(a) Première solution linéaire



(b) Seconde solution linéaire

FIGURE 4.1 – Solutions trouvées pour M_z après de rapides entraînements

calculer notre erreur, en fonction de μ, en remplaçant la somme discrète par une intégrale qui peut être calculée facilement car l'intégrande est alors une fonction polynomiale en $\tilde{t}$:

$$\begin{aligned} E(\mu) &= \frac{1}{t_b - t_a} \int_{t_b}^{t_a} \left(\frac{d\tilde{M}_z}{d\tilde{t}}(\tilde{t}) - \lambda\omega_z(\tilde{M}_z(t)^2 - 1)\right)^2 d\tilde{t} \\ E(\mu) &= \frac{16}{5}\lambda^2\omega_z^2\mu^4 - \frac{8}{3}\lambda\omega_z\mu^3 + (1 - \frac{8}{3}\lambda^2\omega_z^2)\mu^2 + 2\lambda\omega_z\mu + \lambda^2\omega_z^2 \end{aligned} \tag{4.6}$$

On remarque alors que cette fonction admet deux minima locaux qui ont pour valeurs approchées : $\mu_a \approx 0.696$, $E(\mu_a) \approx 3.04$ et $\mu_b \approx -0.574$, $E(\mu_b) \approx -0.78$. Cela est effectivement proche des valeurs que l'on obtient lors des entraînements montrés en figure 4.1. En effet, sur le premier exemple, la sortie du réseau de neurones varie entre 0.60 et 0.74 sur l'intervalle étudié, et la valeur finale de l'erreur est 3.04, ce qui correspond à μ_b et $E(\mu_b)$. Pour le second exemple, la sortie est comprise entre -0.60 et -0.52, et l'erreur vaut 0.82, ce qui correspond bien au second minimum local $E(\mu_a)$.

On peut alors remettre en question le modèle choisi pour notre réseau de neurones, qui n'est pas forcément le plus adapté. On peut par exemple essayer de changer les fonctions d'activation utilisées par les couches intermédiaires, qui étaient des sigmoïdes, et les remplacer par la fonction *relu*.

On a ainsi pu obtenir des résultats satisfaisants visibles en figure 4.2 en lançant l'entraînement avec les fonctions *relu* pour 10000 itérations, avec un taux d'apprentissage de 10^{-3}, $N = 50$, $[t_a = -2, t_b = 2]$. L'erreur finale est de 0.018.

Une autre tentative consiste à simplifier l'architecture du réseau de neurones en supprimant la seconde couche intermédiaire pour ne conserver que

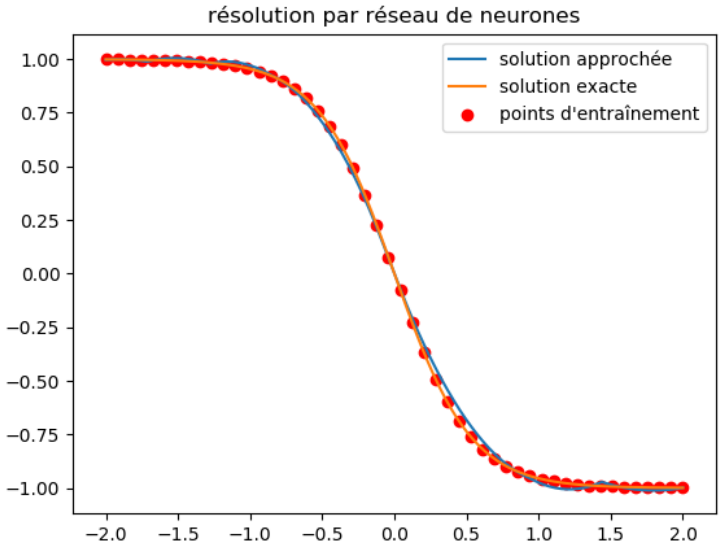


FIGURE 4.2 – Résultats obtenus avec la fonction d'activation *relu*

la première et ainsi diminuer fortement la dimension de l'espace dans lequel on cherche le meilleur vecteur $\boldsymbol{P}$. Avec les paramètres d'entraînement suivants : 100000 itérations, avec un taux d'apprentissage de 1.10^{-2}, $N = 50$, $[t_a = -2, t_b = 2]$. L'erreur finale est de $2.7.10^{-4}$, mais continue à décroître. On voit en figure 4.3 que les courbes des solutions exactes et approchées se confondent.

4.1.2 Résolution des équations sur M_x et M_y

On reprend dans ce paragraphe l'approche de la section 3.2, pour résoudre le système décrivant l'évolution des composantes sur x et sur y, en utilisant pour M_z la solution approchée établie dans la section précédente.

On choisit l'architecture suivante pour le réseau de neurones : 1 entrée pour la variable temporelle, une unique couche intermédiaire de $h_1 = 16$ neurones, utilisant la sigmoïde comme fonction d'activation, et enfin 2 neurones de sortie, avec l'identité comme fonction d'activation en sortie. Ces deux sorties sont notées respectivement $\mathcal{N}_x(t, \boldsymbol{P})$ et $\mathcal{N}_y(t, \boldsymbol{P})$, ce qui nous permet de définir comme précédemment nos fonctions candidates :

$$\begin{cases} \tilde{M}_x(t) = M_{x0} + t\mathcal{N}_x(t, \boldsymbol{P}) \\ \tilde{M}_y(t) = M_{y0} + t\mathcal{N}_y(t, \boldsymbol{P}) \end{cases} \tag{4.7}$$

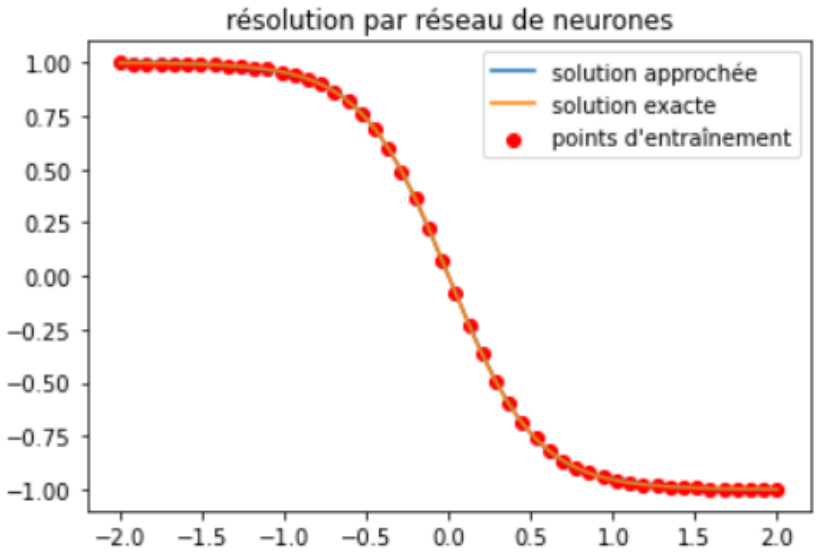


FIGURE 4.3 – Résultats obtenus avec la fonction d'activation *relu*

On définit une fonction d'erreur pour ces solutions potentielles, en s'intéressant aux N points suivants de l'intervalle $[t_a = -1, t_b = 1]$: $\forall i \in [\![1, N]\!], t_i = -1 + 2\frac{i-1}{N}$:

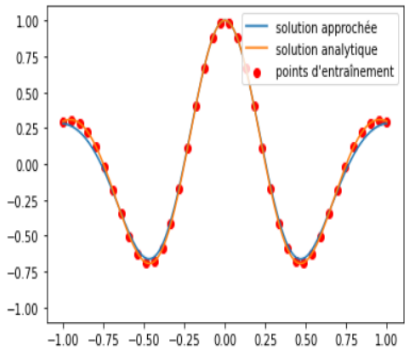
$$\begin{aligned} E(\boldsymbol{P}) = & \frac{1}{N}\sum_{i=1}^{N}\Bigg(\left(\frac{d\tilde{M}_x}{dt}(t_i) + M_y\omega_z + \lambda M_x M_z \omega_z\right)^2 \\ & + \left(\frac{d\tilde{M}_y}{dt}(t_i) - M_x\omega_z + \lambda M_y M_z \omega_z\right)^2\Bigg) \end{aligned} \tag{4.8}$$

On choisit ensuite les paramètres d'entraînement suivants : taux d'apprentissage de 0.01, et 60000 itérations. Les résultats sont regroupés dans la figure 4.4. La solution analytique est représentée en orange, la solution approchée est représentée en bleu, et les points rouges sont les points correspondant aux points d'entraînement $(t_i)_{i\in[\![1,N]\!]}$.

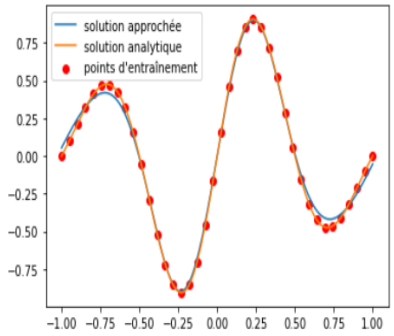
On peut montrer que l'équation 4.1 implique la conservation de la norme de $\boldsymbol{M}$. En effet, en prenant le produit scalaire des deux membres de l'équation par $\boldsymbol{M}$, il vient :

$$\boldsymbol{M} \cdot \frac{d\boldsymbol{M}}{dt} = 0 \Leftrightarrow \frac{d\|\boldsymbol{M}\|^2}{dt} = 0 \tag{4.9}$$

On peut vérifier que cette condition est respectée par le modèle, sachant que la condition initiale impose $\|\boldsymbol{M}\| = 1$. On trace pour cela $\|\boldsymbol{M}\|$ en fonction du temps en figure 4.5, et on constate qu'elle reste comprise entre



(a) Solution obtenue pour M_x



(b) Solution obtenue pour M_y

FIGURE 4.4 – Représentations graphiques de M_x et M_y en fonction du temps par le réseau de neurones

0.975 et 1 sur tout l'intervalle d'étude.

4.2 Résolution sur les trois composantes simultanément

On cherche dans cette section à résoudre directement l'équation 4.1 sans réaliser de projection par le produit vectoriel par $\boldsymbol{M}$. On réécrit l'équation 4.1 comme un système de 3 équations différentielles scalaires, en exprimant simplement l'équation obtenue sur chacune des trois composantes de $\boldsymbol{M}$

$$\begin{cases} \dfrac{dM_x}{dt} = -\omega_z M_y - \lambda(M_z \dfrac{dM_y}{dt} - M_y \dfrac{dM_z}{dt}) \\ \dfrac{dM_y}{dt} = \omega_z M_x - \lambda(M_x \dfrac{dM_z}{dt} - M_z \dfrac{dM_x}{dt}) \\ \dfrac{dM_z}{dt} = -\lambda(M_y \dfrac{dM_x}{dt} - M_x \dfrac{dM_y}{dt}) \end{cases} \tag{4.10}$$

On construit un réseau de neurones similaire à celui utilisé en section 4.1.2, mais en prenant cette fois 32 neurones intermédiaires, et 3 neurones de sortie, de façon à définir les trois expressions suivantes pour les solutions approchées (figure 4.11)

$$\begin{cases} \tilde{M}_x(t) = M_{x0} + t\mathcal{N}_x(t, \boldsymbol{P}) \\ \tilde{M}_y(t) = M_{y0} + t\mathcal{N}_y(t, \boldsymbol{P}) \\ \tilde{M}_z(t) = M_{z0} + t\mathcal{N}_z(t, \boldsymbol{P}) \end{cases} \tag{4.11}$$

over time.PNG

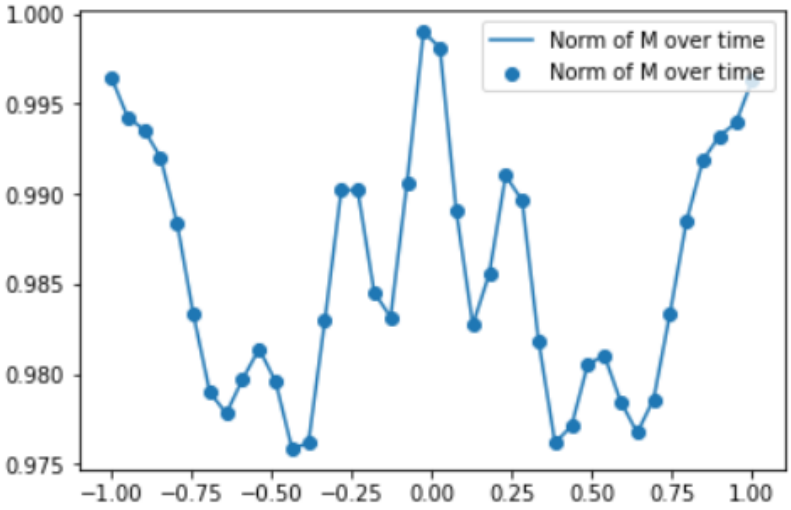
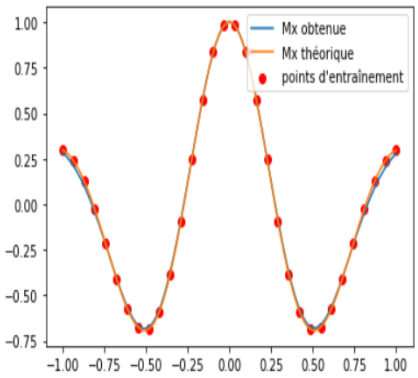


FIGURE 4.5 – Norme de l'aimantation en fonction du temps

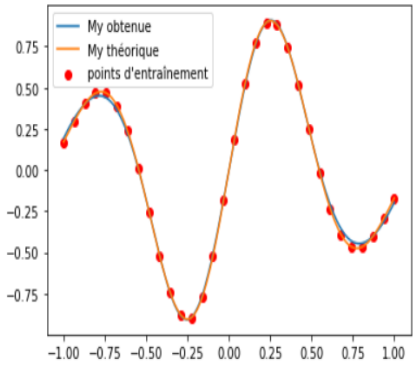
Après entraînement, on trouve les résultats présentés en figure 4.6. Les solutions approchées, représentées en bleu, sont confondues avec les solutions analytiques en orange, établies à partir du système d'équation 4.3.

On peut vérifier comme précédement la conservation de la norme de $\boldsymbol{M}$ au cours du temps, on la trace en figure 4.7. On constate qu'elle reste comprise entre 0.983 et 1.005.

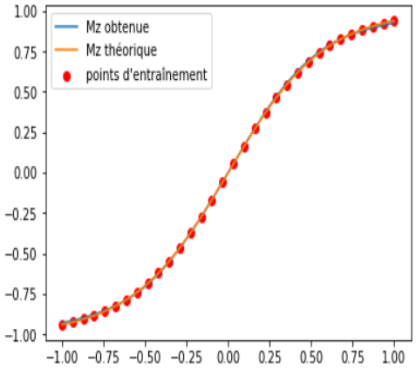
Ainsi, la résolution approchée par réseau de neurones permet de retrouver la solution analytique aussi bien en transformant l'équation par un produit vectoriel par $\boldsymbol{M}$ qu'en la traitant directement.



(a) Solution obtenue pour M_x



(b) Solution obtenue pour M_y



(c) Solution obtenue pour M_z

FIGURE 4.6 – Représentations graphiques de M_x, M_y et M_z en fonction du temps par le réseau de neurones

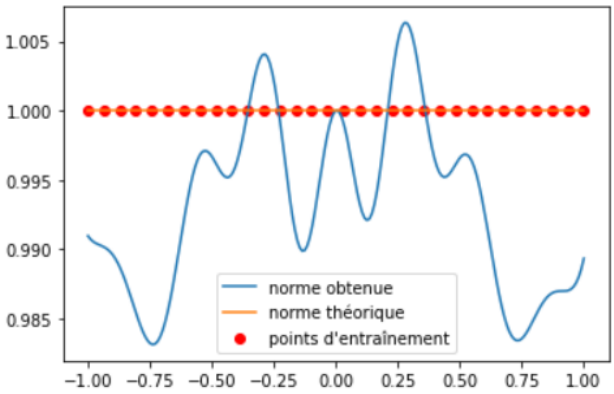


FIGURE 4.7 – Norme de l'aimantation en fonction du temps

Chapitre 5

Conclusion et perspectives

L'étude menée dans le chapitre 2 a permis de comprendre le principe de la résolution d'une équation différentielle par un réseau de neurones, et de montrer que cette approche permettait de résoudre une équation simple à une dimension. Le chapitre 3 a ensuite montré que cette approche donnait également des résultats concluants dans le cas d'un système de deux équations couplées décrivant un mouvement de précession. Cela a également donné l'occasion de prendre en main la librairie TensorFlow afin d'utiliser les méthodes de différentiation automatique pour le calcul de la dérivée de la fonction candidate ainsi que pour celui des gradients de le fonction de coût. Enfin le chapitre 4 a montré que l'approche mise en place permettait de retrouver de façon approchée les solutions analytiques de l'équation de précession avec amortissement.

Il serait utile d'améliorer les permorfances des algorithmes utilisés, en effet les entraînements des réseaux de neurones présentés ici ont rarement atteint une fonction de coût stationnaire. Les entraînements duraient jusqu'à plusieurs dizaines de minutes, et je n'ai pas choisi de les poursuivre plus longtemps lorsque le résultat obtenu était suffisament proche des solutions analytiques. L'amélioration des performances d'entraînement pourrait être réalisée en essayant par exemple des algorithmes d'optimisation plus avancés que la descente de gradients, et également en utilisant d'autres méthodes d'entraînement plus performantes fournies par TensorFlow. On pourrait alors approfondir l'étude des différences de précision selon les différentes architectures de réseaux de neurones pour identifier lesquelles sont les plus pertinentes.

Il serait intéressant d'utiliser l'approche présentée dans ce rapport pour tenter de résoudre l'équation obtenue en remplaçant le terme d'amortissement par un terme de nutation [9], ainsi que l'équation prenant à la fois en compte l'amortissement et la nutation, qui n'admet pas de solution connue.

Bibliographie

[1] "Précession de Larmor" (2022) *Wikipédia*

[2] "Équation de Landau-Lifshitz-Gilbert" (2022), *Wikipédia*

[3] T.T.Dufera, Machine Learning With Applications **5**, 100058 (2021)

[4] I. E. Lagaris, A. Likas and D. I. Fotiadis, IEEE Transactions on Neural Networks **9** no.5, 987-1000 (1998)

[5] A.Malek and R.Shekari Beidokhti, Applied Mathematics and Computation **183**, 260–271 (2006)

[6] Z.Zainuddin and P.Ong, WSEAS Transactions on Mathematics **7** no.6, 333–338 (2008)

[7] "Réseaux de neurones" (2022), *Wikipédia*

[8] "Rayon spectral et suites des puissances" (2022), *Wikipédia*

[9] K. Neeraj *et al.* Nature Physics **17**, 245-250 (2021)