

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

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

Supervisors:
Stam Nicolis
Institut Denis Poisson
Université de Tours, Université d'Orléans, CNRS (UMR7013)
Parc de Grandmont, F-37200, Tours, France
`stam.nicolis@lmpt.univ-tours.fr`

Pascal Thibaudeau
CEA Le Ripault
BP 16, F-37260, Monts, France
`pascal.thibaudeau@cea.fr`

Juillet 2022

Résumé

Faire un résumé de ce qu'il y a dans ce document

Table des matières

1	Introduction	2
2	Equation différentielle d'ordre 1	4
2.1	Solutions en séries de Fourier	4
2.1.1	Inversion du système linéaire	5
2.1.2	Algorithme de descente de gradients	6
2.2	Solutions par réseau de neurones	8
2.2.1	Résultats obtenus	11
3	Mouvement de précession	12
3.1	Solutions en séries de Fourier	12
3.2	Solutions approchées par un réseau de neurones	16
4	Précession avec amortissement	18
4.1	Projection en coordonnées cartésiennes	18
4.1.1	Résolution de l'équation sur M_z	19
4.1.2	Résolution des équations sur M_x et M_y	21
5	Conclusion et perspectives	22

Chapitre 1

Introduction

Ce rapport est le compte-rendu d'un stage d'un mois ayant pour objet la mise en place et l'implémentation de méthodes de résolution d'équations différentielles à l'aide de réseau 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 [2], puis par l'équation de Landau-Lifshitz-Gilbert [1]. Cette seconde version de l'équation ne possède pas de solution analytique connue, 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 à la manière de [3], [4] et [5], en utilisant les algorithmes d'optimisation connus pour minimiser une fonction d'erreur, qui indique dans quelle mesure une fonction candidate, construite à partir du réseau de neurones, est proche d'être solution.

On écrira lorsque cela sera possible l'équation différentielle d'inconnue $\mathbf{M} : \mathbb{R} \rightarrow \mathbb{R}^n$ sous une forme similaire à la suivante, où $\mathbf{g} : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ est une fonction connue :

$$\begin{cases} \frac{d\mathbf{M}(t)}{dt} = \mathbf{g}(t, \mathbf{M}(t)) \\ \mathbf{M}(0) = \mathbf{M}_0 \end{cases} \quad (1.1)$$

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 \mathbf{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 $f : \mathcal{R} \rightarrow \mathcal{R}$ fixée à l'avance (souvent $f = id_{\mathcal{R}}$)

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

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

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

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

Cette partie positionne le travail dans un contexte. Décrire le contexte. Dire ici ce que l'on doit faire et proposer un petit résumé des documents lus de façon à montrer en quoi ils sont pertinents pour le problème posé. Par exemple : Bidulle et Machin dans la référence [?] ont montré que ... tandis que Truc et Chmuc dans la référence [?] ont prouvé que... Dans la section 2, je montrerai que... Dans la section 3, je montrerai... Enfin dans la section 5, je discuterai des résultats obtenus et proposerai quelques perspectives.

Chapitre 2

Equation différentielle d'ordre 1

On souhaite dans ce chapitre se familiariser avec différentes méthodes sur un cas simple avec 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 :

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

On cherche à tester 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 $t = 0$, admet une solution analytique unique qui s'écrit

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

Cela nous permettra par la suite d'évaluer nos solutions numériques, en les comparant à cette solution analytique.

2.1 Solutions en séries de Fourier

On cherche des solutions numériques approchées de l'équation (2.1) sous la forme de séries de Fourier tronquées avec H harmoniques. On écrit pour cela la solution approchée \tilde{M} comme la somme de deux termes, le premier M_0 constant non ajustable vérifiant la condition initiale, et le deuxième $\mathcal{N}(t, \mathbf{A})$

dépendant des coefficients $(A_m)_{m \in \llbracket 1, H \rrbracket}$ représentés par le vecteur \mathbf{A} , construit de façon à ne pas influencer la valeur initiale de la fonction.

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

On définit une fonction d'erreur pour ces solutions potentielles, à partir de la valeur de la dérivée de \tilde{M} aux N points suivants : $\forall i \in \llbracket 1, N \rrbracket, t_i = \frac{i-1}{N}$:

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

On cherche à présent le minimum de E en tant que fonction de \mathbf{A} . Une condition nécessaire sur \mathbf{A} pour être un antécédent d'un minimum est

$$\forall l \in \llbracket 1, H \rrbracket, \frac{\partial E}{\partial A_l} = 0 \quad (2.5)$$

Or ces dérivées partielles sont données pour $l \in \llbracket 1, H \rrbracket$ par :

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

Les deux méthodes suivantes ont pour objectif de trouver les coefficients $(A_m)_{m \in \llbracket 1, H \rrbracket}$ qui vérifient la condition 2.5, et de vérifier que le vecteur de coefficients trouvé correspond bien à la solution analytique, i.e $\forall m \in \llbracket 1, H \rrbracket, A_m = -\frac{1}{2\pi} \delta_1^m$.

2.1.1 Inversion du système linéaire

Dans le cas de cette modélisation pour la fonction approchée, la condition 2.5 correspond à un système linéaire de H équations.

$$\mathcal{M} = (r_{m,l})_{(m,l) \in \llbracket 1, H \rrbracket^2}, \mathbf{A} = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_H \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_H \end{pmatrix} \quad (2.7)$$

$$\forall (m, l) \in \llbracket 1, N \rrbracket^2, \begin{cases} r_{m,l} = 2\pi m l \sum_{i=1}^N \cos(2\pi m t_i) \cos(2\pi l t_i) \\ b_l = -l \sum_{i=1}^N \cos(2\pi t_i) \cos(2\pi l t_i) \end{cases} \quad (2.8)$$

On souhaite alors résoudre le système linéaire en écrivant l'équation matricielle le représentant, c'est-à-dire :

$$\mathcal{M}\mathbf{A} = \mathbf{b} \Leftrightarrow \mathbf{A} = \mathcal{M}^{-1}\mathbf{b} \quad (2.9)$$

On réalise l'implémentation en python, en instanciant les matrices définies précédemment et à l'aide de la bibliothèque *numpy*.

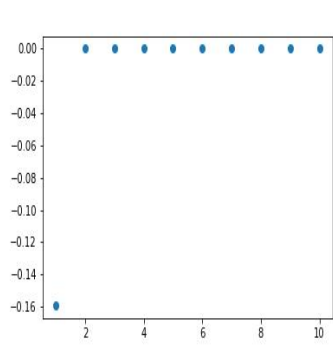
On peut constater que la matrice \mathcal{M} n'est pas toujours inversible selon le choix de H et N . En effet, on peut montrer que c'est nécessairement le cas lorsque $H > N$. Il suffit pour cela d'écrire la matrice \mathcal{M} en fonction de la matrice \mathcal{A} de taille $N \times H$, dont les coefficients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

On remarque également qu'elle ne l'est pas non plus lorsque H et N sont proches par exemple pour $H = N = 10$. Il serait intéressant d'approfondir ce point pour savoir si cela se traduit par le fait que E admette plusieurs minimums locaux par exemple.

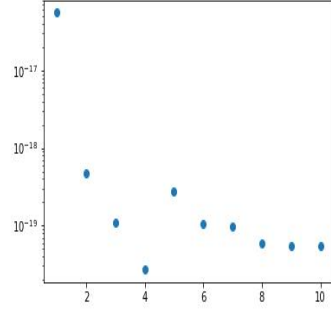
Il semble que choisir $N \gg H$ soit suffisant pour que \mathcal{M} soit inversible. On choisira alors $H = 10, N = 100$ pour la suite. On obtient alors les coefficients présentés en figure 2.1 avec les valeurs absolues des erreurs de chacun par rapport à la valeur théorique. On constate que les erreurs sur chaque coefficient est inférieure à 10^{-16} , on valide donc cette première méthode.

2.1.2 Algorithme de descente de gradients

On définit les paramètres suivants :



(a) Coefficients trouvés



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

FIGURE 2.1 – Résultats de la méthode de l'inversion de système

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

Puis on calcule itérativement :

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

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 \mathbf{b} définis précédemment qui sont indépendants de \mathbf{A} et de k :

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

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

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

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 $\mathcal{R}_\alpha =$

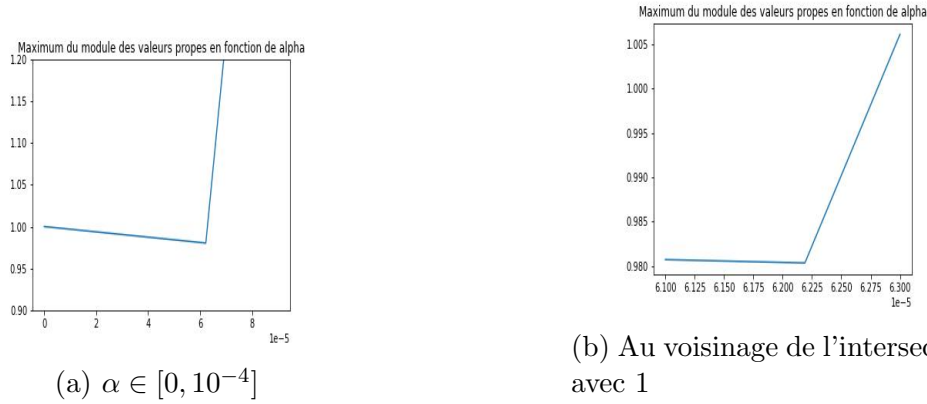


FIGURE 2.2 – Rayon spectral de \mathcal{R}_α en fonction de α

$(\mathcal{I}_H - \alpha\mathcal{M})$, noté $\rho(\mathcal{R}_\alpha)$. La suite $(\mathbf{A}^{(n)})_{n \in \mathbb{N}}$ converge si et seulement si la norme de $(\mathcal{R}_\alpha^n)_{n \in \mathbb{N}}$ tend vers 0. 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 $\alpha_c = 6.2807 \cdot 10^{-5}$, pour laquelle $\rho(\mathcal{R}_\alpha) = 1$, ainsi que la valeur $\alpha_{min} = 6.2189 \cdot 10^{-5}$ pour $\rho(\mathcal{R}_\alpha)$ est minimal.

On exécute l'algorithme en parallèle pour les deux valeurs de α trouvées précédemment ainsi que pour une valeur $\alpha_1 = 6.3 \cdot 10^{-5}$, 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. Tandis que 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, elle 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} , on valide donc cette seconde méthode.

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

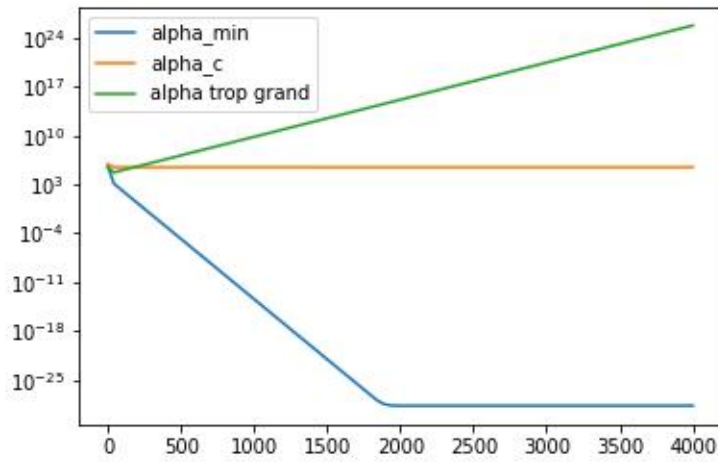
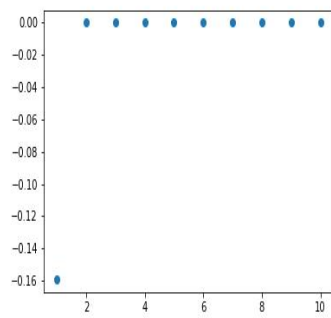
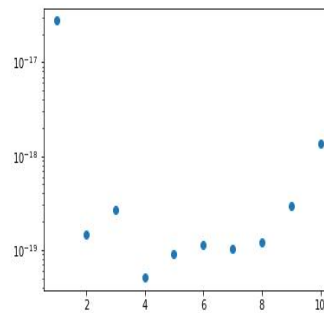


FIGURE 2.3 – Erreurs en fonction du nombre d'itérations pour 3 valeurs de α



(a) Coefficients trouvés



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

FIGURE 2.4 – Résultats de la méthode de descnte de gradients

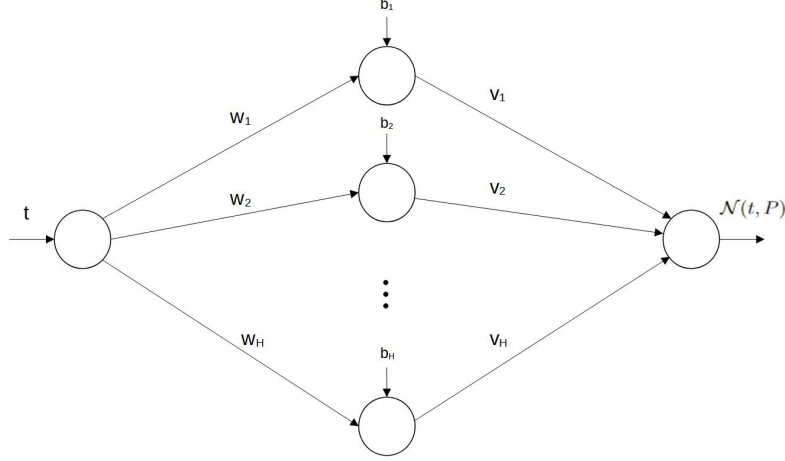


FIGURE 2.5 – Réseau de neurones

approchées sous la forme suivante :

$$\begin{cases} \tilde{M}(t) = M_0 + t\mathcal{N}(t, \mathbf{P}) \\ \mathcal{N}(t, \mathbf{P}) = \sum_{j=1}^H v_j \sigma(w_j t + b_j) \end{cases} \quad (2.14)$$

$\mathcal{N}(t, \mathbf{P})$ 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 $\forall x \in \mathbf{R}, \sigma(x) = \frac{1}{1 + e^{-x}}$. Les paramètres \mathbf{P} à ajuster sont désormais les coefficients $(w_j)_{j \in \llbracket 1, H \rrbracket}$, $(b_j)_{j \in \llbracket 1, H \rrbracket}$ et $(v_j)_{j \in \llbracket 1, H \rrbracket}$.

Peux-tu donner une référence pour quelqu'un qui cherche ce que tout ceci veut dire ?

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

$$E(\mathbf{P}) = \sum_{i=1}^N \left(\frac{d\tilde{M}}{dt}(t_i) + \cos(2\pi t_i) \right)^2 \quad (2.15)$$

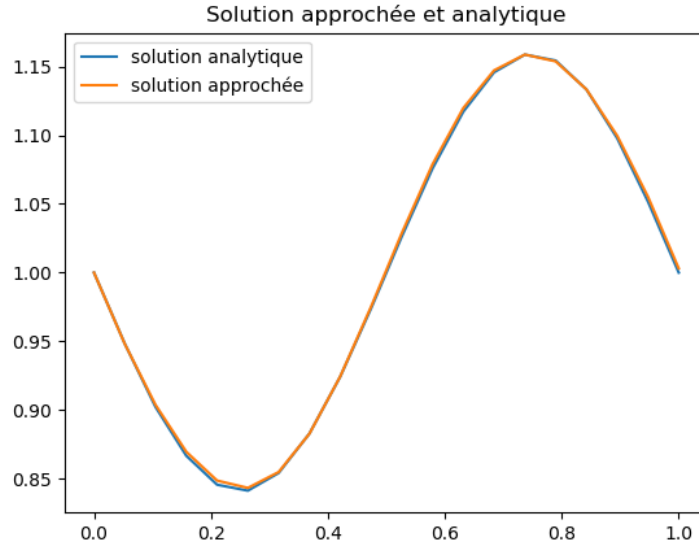


FIGURE 2.6 – estimation de la solution par un réseau de neurones

L'équation (2.15) est-elle correcte ?

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

2.2.1 Résultats obtenus

On initialise l'algorithme avec les paramètres suivants : ($H = 4, N = 20$)
On obtient une erreur de $1,2 \cdot 10^{-2}$ et une estimation visible en figure 2.6.
Cela permet de valider notre modèle sur l'étude à une dimension.

Chapitre 3

Mouvement de précession

On s'intéresse désormais au problème de la précession d'un moment magnétique dans un champ magnétique constant. On le modélise par les équations suivantes pour $t \in [t_a = 0, t_b = 1]$:

$$\begin{cases} \frac{dM_x}{dt} = \omega M_y \\ \frac{dM_y}{dt} = -\omega M_x \end{cases} \quad (3.1)$$

avec les conditions initiales

$$\begin{cases} M_x(0) = M_0 \\ M_y(0) = 0 \end{cases} \quad (3.2)$$

dont la solution analytique vaut

$$\begin{cases} M_x(t) = M_0 \cos(2\omega t) \\ M_y(t) = -M_0 \sin(2\omega t) \end{cases} \quad (3.3)$$

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 :

$$\begin{cases} \tilde{M}_x(t) = M_0 + \sum_{m=1}^H A_m (\cos(m\omega t) - 1) + B_m \sin(m\omega t) \\ \tilde{M}_y(t) = \sum_{m=1}^H -A_m \sin(m\omega t) + B_m (\cos(m\omega t) - 1) \end{cases} \quad (3.4)$$

Les coefficients $(A_m)_{m \in \llbracket 1, H \rrbracket}$ et $(B_m)_{m \in \llbracket 1, H \rrbracket}$ sont les paramètres à ajuster. On cherche à obtenir la solution analytique, i.e $\forall m \in \llbracket 1, H \rrbracket, A_m = \delta_1^m$ et $\forall m \in \llbracket 1, H \rrbracket, B_m = 0$.

On définit une fonction d'erreur pour ces solutions potentielles, en s'intéressant aux N points suivants : $\forall i \in \llbracket 1, N \rrbracket, t_i = \frac{i-1}{N}$:

$$E(\mathbf{P}) = \sum_{i=1}^N \left(\frac{d\tilde{M}_x}{dt}(t_i) - \omega \tilde{M}_y(t_i) \right)^2 + \left(\frac{d\tilde{M}_y}{dt}(t_i) + \omega \tilde{M}_x(t_i) \right)^2 \quad (3.5)$$

On utilise ensuite la méthode de descente de gradients définie précédemment, en calculant les dérivées partielles suivantes : $(\frac{\partial E}{\partial A_l}, \frac{\partial E}{\partial B_l})_{l \in \llbracket 1, H \rrbracket}$

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

$$\mathbf{P}^{(k)} = \begin{pmatrix} A_1^{(k)} \\ \vdots \\ A_H^{(k)} \\ B_1^{(k)} \\ \vdots \\ B_H^{(k)} \end{pmatrix}, \mathbf{g}^{(k)} = \begin{pmatrix} \frac{\partial E^{(k)}}{\partial A_1^{(k)}} \\ \vdots \\ \frac{\partial E^{(k)}}{\partial A_H^{(k)}} \\ \frac{\partial E^{(k)}}{\partial B_1^{(k)}} \\ \vdots \\ \frac{\partial E^{(k)}}{\partial B_H^{(k)}} \end{pmatrix}, \quad (3.6)$$

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

$$\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \alpha \mathbf{g}^{(k)} \quad (3.7)$$

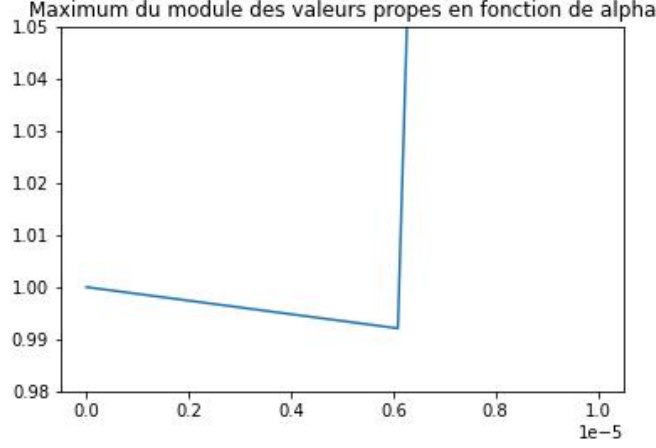


FIGURE 3.1 – Evolution du maximum du module en fonction de α

$$\mathbf{g}^{(k)} = \mathcal{M}\mathbf{P}^{(k)} - \mathbf{b} \quad (3.8)$$

$$\mathbf{P}^{(k+1)} = (\mathcal{I}_H - \alpha\mathcal{M})\mathbf{P}^{(k)} + \alpha\mathbf{b} \quad (3.9)$$

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 est supérieur à 1 et la suite diverge. On trouve les valeurs suivantes $\alpha_{min} = 6.0906 \cdot 10^{-6}$ et $\alpha_c = 6.1146 \cdot 10^{-6}$, et l'évolution du maximum en fonction de α est montré en figure 3.1.

On exécute 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 \mathbf{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} .

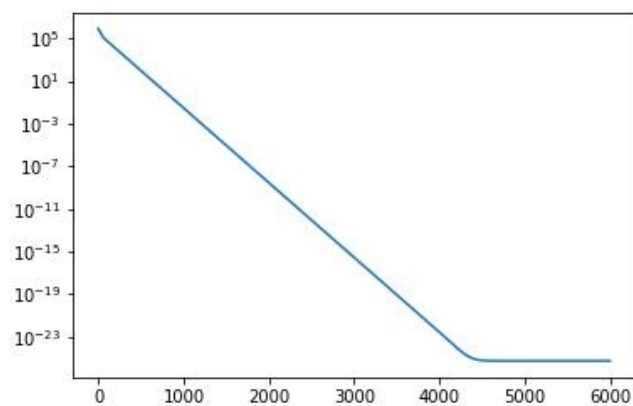
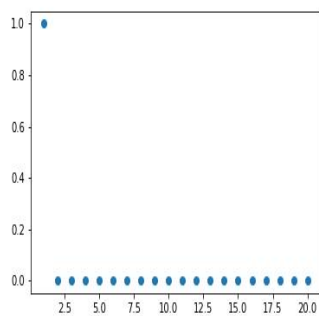
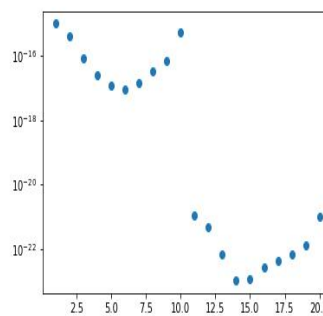


FIGURE 3.2 – Valeur de la fonction d'erreur en fonction du nombre d'itérations



(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 approchées par un réseau de neurones

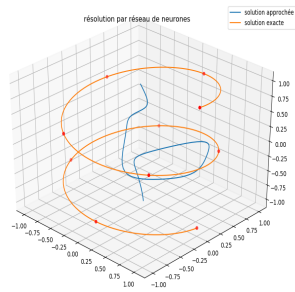
On cherche dans ce paragraphe à approcher la solution du système des 2 équations grâce à un réseau de neurones. On choisit l'architecture suivante pour le réseau de neurones : 1 entrée pour la variable temporelle, une première couche intermédiaire de $h_1 = 32$ neurones, une seconde couche intermédiaire de $h_2 = 8$ neurones, toutes deux utilisant la sigmoïde définie précédemment 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 $\mathcal{N}_x(t, \mathbf{P})$ et $\mathcal{N}_y(t, \mathbf{P})$, où \mathbf{P} représente l'ensemble des paramètres du réseau de neurones. On construit alors nos solutions approchées de la façon suivante :

$$\begin{cases} \tilde{M}_x(t) = M_0 + t\mathcal{N}_x(t, \mathbf{P}) \\ \tilde{M}_y(t) = t\mathcal{N}_y(t, \mathbf{P}) \end{cases} \quad (3.10)$$

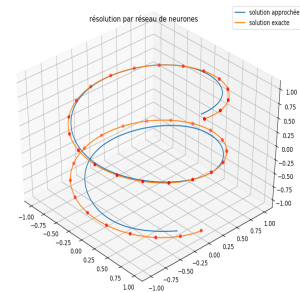
On définit une fonction d'erreur pour ces solutions potentielles, en s'intéressant aux N points suivants de l'intervalle $[t_a = -1, t_b = 1] : \forall i \in \llbracket 1, N \rrbracket, t_i = -1 + 2\frac{i-1}{N} :$

$$E(\mathbf{P}) = \frac{1}{N} \sum_{i=1}^N \left(\frac{d\tilde{M}_x}{dt}(t_i) - \omega \tilde{M}_y(t_i) \right)^2 + \left(\frac{d\tilde{M}_y}{dt}(t_i) + \omega \tilde{M}_x(t_i) \right)^2 \quad (3.11)$$

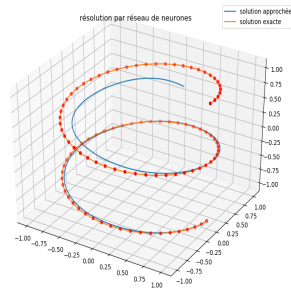
On choisit ensuite les paramètres d'entraînement suivants : taux d'apprentissage : 0.007, 50000 itérations, et on lance 3 entraînements pour $N \in \{10, 40, 100\}$. On obtient les résultats montrés en figure 3.4, où l'axe vertical représente le temps, et les axes horizontaux représentent M_x et M_y . 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 $(t_i)_{i \in \llbracket 1, N \rrbracket}$.



(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 $\mathbf{M} \in \mathbb{R}^3$ 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 $\boldsymbol{\omega} = \omega_z \mathbf{z}$.

L'équation considérée prend le nom d'équation de précession avec un terme d'amortissement introduit par Gilbert [1] et s'écrit

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

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

4.1 Projection en coordonnées cartésiennes

En prenant le produit vectoriel par \mathbf{M} à gauche des deux membres de l'équation 4.1, on forme une projection sur un plan normal à \mathbf{M} , et on obtient 3 nouvelles équations différentielles scalaires (après un chagement d'échelle

temporelle) :

$$\begin{cases} \frac{dM_x}{d\tilde{t}} = -M_y\omega_z - \lambda M_x M_z \omega_z \\ \frac{dM_y}{d\tilde{t}} = M_x\omega_z - \lambda M_y M_z \omega_z \\ \frac{dM_z}{d\tilde{t}} = \lambda\omega_z(1 - M_z^2) \end{cases} \quad (4.2)$$

On remarque que l'équation sur M_z est découplée des 2 autres, on cherche ainsi à résoudre celle-ci en premier lieu.

4.1.1 Résolution de l'équation sur M_z

On utilise la même approche avec un réseau de neurones que précédemment à l'aide d'un réseau de neurones. On réalisera à présent les implémentations en python à l'aide de la bibliothèque TensorFlow.

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 \mathbf{P} . Comme précédemment, afin de satisfaire la condition initiale, on cherche la fonction \tilde{M}_z sous la forme suivante, où $\mathcal{N}(\tilde{t}, \mathbf{P})$ représente la sortie du réseau de neurones.

$$\tilde{M}_z(\tilde{t}) = M_z(0) + \tilde{t}\mathcal{N}(\tilde{t}, \mathbf{P}) \quad (4.3)$$

On définit la fonction d'erreur suivante :

On fixe les paramètres suivants : $\lambda = 0.3$, nombre de points de tests : $N = 30$, 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 minimiser la fonction d'erreur dans le cas où M_z est linéaire par rapport à \tilde{t} , (d'après l'équation 4.3, pour $\mathcal{N}(\tilde{t}, \mathbf{P}) = \mu \in \mathbf{R}$ 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 calculer notre erreur, en fonction de μ , en remplaçant la somme discrète par

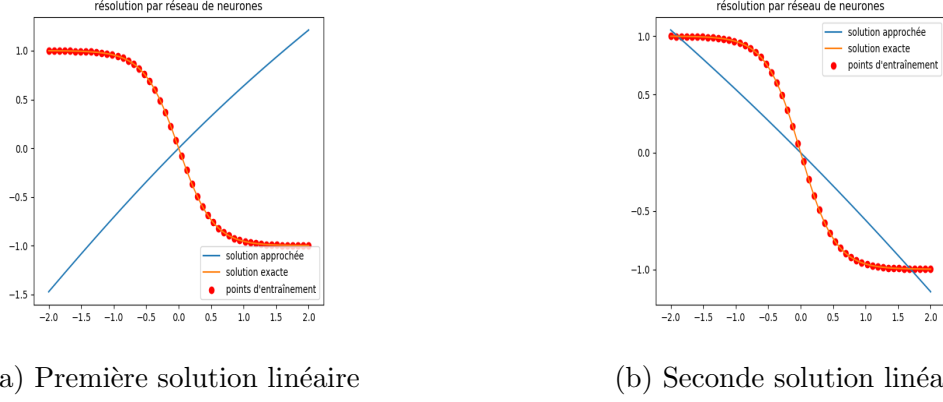


FIGURE 4.1 – Solutions trouvées pour M_z après de rapides entraînements

une intégrale qui peut être calculée facilement car l'intégrande est alors une fonction polynomiale en \tilde{t} :

$$\begin{aligned}
 E(\mu) &= \frac{1}{t_b - t_a} \int_{t_b}^{t_a} \left(\frac{d\tilde{M}_z}{d\tilde{t}}(\tilde{t}) - \lambda\omega_z(\tilde{M}_z(\tilde{t})^2 - 1) \right)^2 d\tilde{t} \\
 E(\mu) &= \frac{16}{5}\lambda^2\omega_z^2\mu^4 - \frac{8}{3}\lambda\omega_z\mu^3 + \left(1 - \frac{8}{3}\lambda^2\omega_z^2\right)\mu^2 + 2\lambda\omega_z\mu + \lambda^2\omega_z^2
 \end{aligned} \tag{4.4}$$

On remarque alors que cette fonction admet deux minimums 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} , $N = 50$, $[t_a = -1, t_b = 1]$. L'erreur finale est de 0.018.

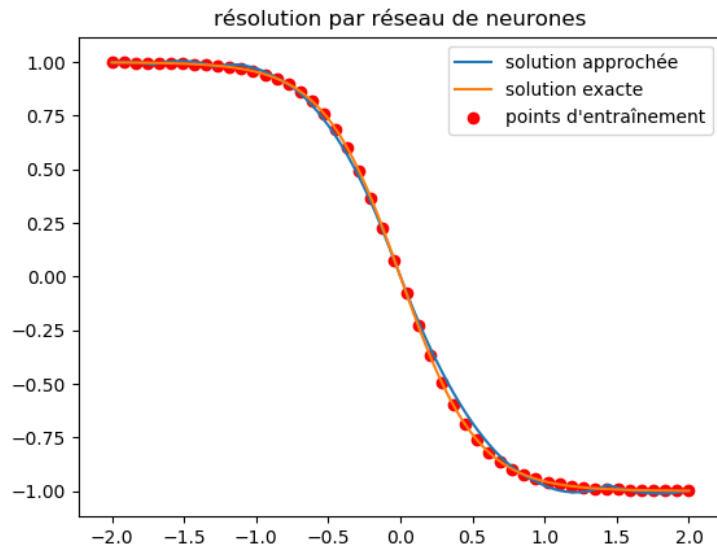


FIGURE 4.2 – Résultats obtenus avec la fonction d'activation *relu*

Ainsi, la fonction *relu* semble faciliter la convergence du réseau de neurones.

4.1.2 Résolution des équations sur M_x et M_y

Chapitre 5

Conclusion et perspectives

Bibliographie

- [1] Wikipedia, Équation de Landau-Lifshitz-Gilbert
- [2] Wikipedia, Précession de Larmor
- [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 pp. 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 Volume **7** Issue 6 pp 333–338(2008)
- [7] Wikipedia, Réseaux de neurones
- [8] Wikipedia, Rayon spectral et suites des puissances