Méthodes de Machine Learning pour les équations différentielles stochastiques

Dorian Coudiere

Mai 2024

Contents

1 Introduction

La création et l'analyse de schéma numérique constituent un champ de recherche actif. Ces dernières décennies, la théorie des équations modifiées a pris de l'ampleur comme un outil majeure d'analyse et d'amélioration de schéma numériques. Soit une équation différentielle ordinaire (EDO) de la forme :

$$\dot{y} = f(y(t)) \in \mathbb{R}^d, 0 \leq t \leq T$$

où $f : \mathbb{R}^d \to \mathbb{R}^d$ est suffisamment lisse auquel on associe le flow exact φ_h^f. Une méthode numérique définit une approximation ϕ_h^f du flot exact pour un pas h à l'ordre r telle que

$$\phi_h^f = \varphi_h^f + O(h^{r+1})$$

L'idée est de trouver une équation modifiée $\dot{y} = \tilde{f}(y(t))$ telle que

$$y_{n+1} = \phi_h^{\tilde{f}}(y_n) = \varphi_h^f(y_n)$$

On écris le champs modifié sous la forme d'une B-série [4]

$$\tilde{f}(y) = f(y) + \sum_{i=1}^{\infty} h^i f_i(y)$$

La série ne converge pas toujours, mais l'idée est de tronquer la série à l'ordre N afin d'avoir un intégrateur d'ordre N+1. C'est une théorie très intéressante, car en plus d'améliorer l'ordre, l'intégrateur conserve certaines propriétés géométriques sur un temps long. Cette théorie peut être plus ou moins bien étendue au cas des équations différentielles stochastiques, mais on s'intéressera ici au calcul des coefficients dans le cas stochastique. En s'inspirant de travaux réalisés dans le cas déterministe [2] on étudiera une méthode novatrice basée sur du Machine Learning afin de déterminer l'expression de ces coefficients.

2 Intégration géométrique stochastique

2.1 Rappel sur les EDS

Soit $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{P})$ un espace de probabilité complet filtré, $f : \mathbb{R}^d \to \mathbb{R}^d$ et $\sigma : \mathbb{R}^d \to \mathbb{R}^{d\times m}$ des fonctions suffisamment lisses et Lipschitz continue. Une équation différentielle stochastique (EDS) sous forme d'Ito peut s'écrire de la façon suivante

$$dX = f(X)dt + \sigma(X)dW(t) \tag{2.1}$$

où $X(0) = x \in \mathbb{R}^d$ et $W(.) = (W_1(t), ..., W_m(t))^T$ sont m processus standard de Wiener indépendants. Dans la suite on prendra $m = 1$.

On peut par analogie avec les EDO définir des schémas numériques à un pas pour les EDS afin d'approximer (2.1)

$$Y_{n+1} = \phi_h^{f,\sigma}(Y_n) \tag{2.2}$$

On dit qu'un schéma est d'ordre faible r si pour chaque fonction test ϕ,pour tout $T \geq 0$, $T = Nh$, h assez petit et $X_0 = x$ il existe une constante positive $C(\phi, x)$ telle que

$$|\mathbb{E}[\phi(X_N)] - \mathbb{E}[\phi(X(T))]| \leq C(\phi, x)h^r \tag{2.3}$$

On introduit un outils pour l'étude des EDS, le générateur,que l'on peut définir de façon formelle [1] mais qu'on définira pour (2.1) comme

$$\mathcal{L}\phi = \phi' f + \frac{\sigma^2}{2}\phi'' \tag{2.4}$$

Sous certaines hypothèses de régularité, $u(x,t) = \mathbb{E}[\phi(X(t))]$, satisfait l'équation dit "Backward Kolmogorov"

$$\frac{\partial u}{\partial t}(x,t) = \mathcal{L}u(x,t), u(x,0) = \phi(x), t > 0 \tag{2.5}$$

Proposition 1. *Pour h suffisament faible,* (2.5) *nous permet d'en déduire*

$$u(x,h) = \phi(x) + \sum_{j=1}^{N} \frac{h^j}{j!}\mathcal{L}^j\phi(x) + h^{N+1}R_N^h(\phi, x) \tag{2.6}$$

Proposition 2. *[1] Pour toute fonction test ϕ, l'intégrateur numérique à une expansion de Taylor faible de la forme :*

$$E[\phi(X_1)] = \phi(x) + \sum_{j=1}^{N} h^j \mathcal{A}_{j-1}\phi(x) + h^{N+1}R_N^h(\phi, x) \tag{2.7}$$

pour h assez faible et x chosis dans un bon domaine.

Proposition 3. *(Talay et Tubaro 90) Sous certaines conditions, si*

$$\mathcal{A}_{j-1} = \frac{\mathcal{L}^j}{j!}, j = 1, ..., r, \tag{2.8}$$

alors le schéma numérique est au moins d'ordre faible r

2.2 Modified Equations

L'idée est d'étendre la notion d'équations modifiées aux équations différentielles stochastiques en construisant $\tilde{f}$ et $\tilde{\sigma}$ comme des B-séries [4],

$$\begin{cases} \tilde{f}_h(y) = f(y) + \sum_{i=1}^{\infty} h^i f_i(y) \\ \tilde{\sigma}_h(y) = \sigma(y) + \sum_{i=1}^{\infty} h^i \sigma_i(y) \end{cases} \tag{2.9}$$

Shardlow [5] puis plus tard Zygalakys ont essayer de calculer les différents coeficients de façon générale en utilisant les outils de la partie précédente. On va reprends le calcul en troncquant à l'ordre 1 et en utilisant comme schéma numérique Euler-Maruyama, (2.2) devient

$$X_1 = x + h\tilde{f}(x) + \sqrt{h}\tilde{\sigma}(x)\xi \tag{2.10}$$
$$= x + h(f + hf_1) + \sqrt{h}(\sigma + h\sigma_1)\xi \tag{2.11}$$

et donc (2.7) devient

$$\begin{aligned} E[\phi(\tilde{X_1})] &= \phi(x) + h(\frac{\sigma}{2}\phi'' + \phi' f) + h^2(\phi' a_1 + \frac{f^2}{2}\phi'' + \phi''\sigma\sigma_1 + \frac{1}{8}\sigma^4\phi^{(4)} + \frac{1}{2}f\sigma^2\phi^{(3)}) \\ &= \phi(x) + L\phi(x) + A_1\phi''(x) + ... \end{aligned}$$

On en déduit :

$$\begin{aligned} \frac{L^2\phi}{2} - \tilde{A}_1\phi &= \phi'(\frac{1}{2}f'f + \frac{1}{4}f''\sigma^2 - f_1) \\ &\quad + \phi''(-\frac{1}{2}f^2 - \sigma\sigma_1 + \frac{1}{2}\sigma^2 + \frac{1}{4}\sigma'^2\sigma^2 + \frac{1}{2}f\sigma'\sigma + \frac{1}{2}f'\sigma^2 + \frac{1}{2}\sigma'\sigma^3) + \phi^{(3)}(\frac{1}{2}\sigma'\sigma^3) \\ &= 0 \end{aligned}$$

Il n'est pas possible de trouver f_1 et σ_1 afin d'assurer la condition de Talay Tubaro donc on ne peut pas trouver d'expression générale. Cependant cela marche sur des cas spécifiques.

2.3 Etude des cas spécifiques

2.3.1 EDS Linéaire

On choisis un premier exemple linéaire, on prends un problème stochastique de la forme,

$$dY = \lambda Y dt + \mu Y dW, \lambda \in \mathbb{R}, \mu \in \mathbb{R} \tag{2.12}$$

Ce problème est un problème standard dont on connait l'unique solution, une Variable aléatoire dépendant d'un bruit Brownien dont on peut calculer l'espérance et la variance.

$$Y(t) = e^{(\lambda - \frac{\mu^2}{2})t + \mu W(t)} Y_0 \tag{2.13}$$

On chosis comme Schéma numérique Euler-Maruyama que l'on réécrira comme

$$Y_1 = Y_0 + h\tilde{f}(x) + \sqrt{h\tilde{\sigma}^2}\xi \tag{2.14}$$

avec $\tilde{\sigma}^2$ qui peut aussi s'écrire sous forme d'une B-série. De plus, on tronque les séries à l'ordre 2. D'après (2.13) et (2.14), on a

$$\begin{aligned} E[Y(h)] &= (1 + \lambda h + \frac{(\lambda h)^2}{2} + \frac{(\lambda h)^3}{6} + O(h^4))Y_0 \\ &= Y_0 + hf(Y_0) + h^2 f_1(Y_0) + h^3 f_2(Y_0) + O(h^4) \end{aligned}$$

$$\begin{aligned} Var[Y(h)] &= (\mu^2 h + (2\lambda\mu^2 + \frac{\mu^4}{2})h^2 + (\frac{\mu^6}{6} + 2\lambda^2\mu^2 + \mu^4\lambda)h^3 + O(h^4))Y_0^2 \\ &= h\sigma^2(Y_0) + h^2\sigma_1^2(Y_0) + h^3\sigma_2^2(Y_0) + O(h^4) \end{aligned}$$

On en déduit immédiatement :

$$\begin{cases} f_1(x) = & \frac{\lambda^2}{2}x \\ \sigma_1^2(x) = & (\frac{\mu^4}{2} + 2\lambda\mu^2)x^2 \\ f_2(x) = & \frac{\lambda^3}{6}x \\ \sigma_2^2(x) = & (\frac{\mu^6}{6} + \lambda\mu^4 + 2\lambda^2\mu^2)x^2 \end{cases}$$

2.3.2 Pendule Stochastique

On décide ensuite d'étudier un problème plus complexe, un problème non-linéaire, on va étudier un système Hamiltonien, un pendule stochastique. On considère

$$\begin{cases} H(q,p) = \frac{p^2}{2} + \sin(q) \\ f(q,p) = J\nabla H(q,p) = \begin{pmatrix} p \\ -\sin(q) \end{pmatrix} \end{cases} \tag{2.15}$$

On construit l'EDS associée sous forme Stratonovich

$$dX = f(q,p)(dt + \circ dW) \tag{2.16}$$

Ce problème est intéressant en plus de sa non linearité d'une part car on considère Y de dimension 2, cela va donc augmenter le nombre de neuronnes dans les couches d'entrée et de sortie et donc complexifier l'apprentissage de notre réseaux. Mais surtout car ce système est Hamiltonien. On va donc chosir comme méthode de base point milieu stochastique qui est une méthode symplectique et que l'on va essayer d'améliorer d'un ordre. On peut calculer les différents coefficients en utilisant les propositions en partie 2.1, et on obtient :

$$f_1 = \frac{1}{8}(f''(f,f) - 2f'f'f) = \begin{pmatrix} \frac{1}{4}cos(q)p \\ \\ \frac{1}{8}(sin(q)p^2 - 2cos(q)sin(q)) \end{pmatrix} \tag{2.17}$$

3 Des réseaux de neuronnes pour les équations modifiés

3.1 Stratégie Géneral

On va chercher à construire un réseau de neurones qui approxime les champs modifiés (2.9), on décide de garder la structure du champ modifié, on construit donc les fonctions approximées de la forme

$$\begin{cases} f_{app}(y,h) = f(y) + hf_1(y) + ... + h^q R_1(y,h) \\ \sigma_{app}(y,h) = \sigma(y) + h\sigma_1(y) + ... + h^q R_2(y,h) \end{cases} \tag{3.1}$$

de sorte que l'erreur faible dans le sens (2.3) soit d'ordre q. On va approximer chaque terme par un réseau de neurones plutôt que le tout par un seul pour assurer la consistance de la méthode. Chacun des termes des 2 B-séries tronquées ainsi que les restes sont représentés par des Multi layer perceptron (MLP) , composés d'un nombre de couches cachées et d'une fonction d'activation variable.

On construit ensuite notre dataset de la façon suivante, on choisit K différents $y_0^{(i)}$ de façon uniforme dans un compact représantatif du domaine de simulation, on choisit aussi K différents $h^{(i)}$, en choisissant $\log(h^{(i)})$ de façon uniforme dans $[\log(h_-), \log(h_+)]$, puis on calcul pour chaque point N trajectoires $y_1^{(i)}(\omega)$, d'une variable aléatoire $y_1^{(i)}$ aussi proches possibles de $\varphi_{h^{(i)}}^{f,\sigma}(y_0^{(i)})$. Pour cela on choisit la méthode de base et on l'applique avec un pas assez petit. On calcule ensuite une estimation de l'espérance $E[y_1^{(i)}]$, par un estimateur de Monte-Carlo

$$\bar{X} = \frac{1}{N}\sum_{k=1}^{N} X_k \tag{3.2}$$

puis la matrice de covariance $Cov[y_1^{(i)}]$, par un estimateur

$$\mathrm{Cov}(x,y) = \frac{\sum_{i=1}^{N}(X_i - \bar{X})(Y_i - \bar{Y})}{max(0, N - \delta N)} \tag{3.3}$$

On va ensuite entraîner notre modèle, pour chaque point k , on simule N trajectoires en utilisant les valeurs courantes des MLP du flot modifié du système appliqué à la condition $y_0^{(i)}$ au pas de temps $h^{(i)}$, $\phi_{h^{(k)}}^{f_{app}(.,h^{(k)}),\sigma_{app}(.,h^{(k)})}(y_0^{(k)})$. On estime ensuite l'espérance et la covariance de la même manière que pour la création de nos données. Il faut ensuite estimer l'erreur entre les données prédites et les données réelles précédemment approchées. Contrairement à ce qui a été fait dans le cas déterministe [2] , on ne peut pas calculer directement la quantité aléatoire $|\phi_{h^{(k)}}^{f_{app}(.,h^{(k)}),\sigma_{app}(.,h^{(k)})}(y_0^{(k)}) - \varphi_{h^{(k)}}^{f,\sigma}(y_0^{(k)})|$, on construit donc notre fonction de perte comme

$$Loss_{train} = \frac{1}{K}\sum_{K=0}^{K-1}\left[\frac{|E[\phi_{h^{(k)}}^{f_{app}(.,h^{(k)}),\sigma_{app}(.,h^{(k)})}(y_0^{(k)})] - E[\varphi_{h^{(k)}}^{f,\sigma}(y_0^{(k)})]|}{h^{(k)p+1}}\right. \tag{3.4}$$

$$\left. + \frac{|Cov[\phi_{h^{(k)}}^{f_{app}(.,h^{(k)}),\sigma_{app}(.,h^{(k)})}(y_0^{(k)})] - Cov[\varphi_{h^{(k)}}^{f,\sigma}(y_0^{(k)})]|}{h^{(k)p+1}}\right] \tag{3.5}$$

Remarque 1. *L'étude des 2 premiers moments ne suffisent pas à déduire la loi de notre Variable cependant ils suffisent dans nos cas à trouver les différents termes des B-séries, c'est pourquoi on se limite à l'espérance et la covariance dans notre fonction de perte.*

On met en suite à jour les poids et les biais de nos différents MLP pour minimiser cette fonction de perte. Afin d'évaluer la performance de notre apprentissage, on utilise de plus une stratégie classique, on divise notre ensemble de données en 2, 80% qui est utilisé aussi pour l'entraînement et donc sur lequel on évalue la fonction de perte sur des données connues et 20% qui est réservée à l'évaluation et qui ne sera jamais donnée à notre réseau de neurones afin d'observer les potentiels phénomènes de surapprentissage.

3.2 Réglage des hyperparamètres

Maintenant que notre modèle et notre stratégie d'apprentissage bien définit, il reste néanmoins à calibrer le modèle, ce qui consiste en le réglage des hyperparamètres.

Les hyperparamètres sont des paramètres définies en amont de l'apprentissage contrairement aux poids par exemples qui eux varient et qui vont fortement influer sur la vitesse et la qualité de celui-ci. On va donc bien régler ceux-ci. On retrouve parmi les principaux hyperparamètres :

- Nombre de couches cachées et nombre de neuronne par couche cachée

  Ce sont les paramètres géométriques du réseau de neurones. Le théorème d'approximation universel nous dit que l'on peut approximer n'importe quel fonction continue définit sur un compact grâce à un MLP avec seulement 1 couche cachée. Cependant, le théorème ne nous dit rien sur le nombre de neurones sur cette couche et pour des fonctions complexes, il peut exploser. On va donc fixer le nombre de neurones et tester avec plusieurs profondeurs.

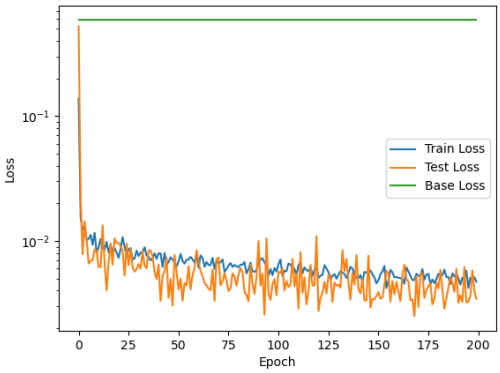

Figure 1: Fonction de perte sur le cas Linéaire en utilisant des MLPs avec 5 couches cachées et 40 neurones par couche

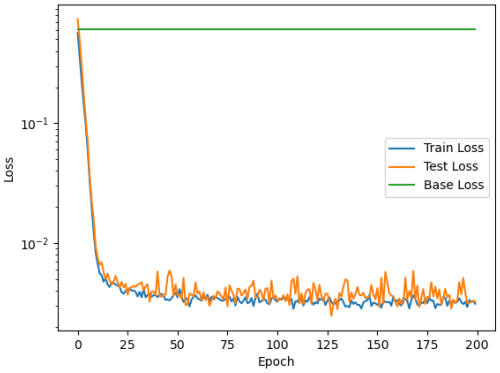

Figure 2: Fonction de perte sur le cas Linéaire en utilisant des MLPs avec 1 couches cachées et 100 neurones par couche

On observe que pour un réseau de neurones plus profonds, la perte va converger vers une meilleure valeur et en moins d'epochs et donc très probablement fournir de meilleurs résultats. Cependant, avoir un réseau de neurones trop profonds peut créer des problèmes comme l'exploding gradient, le vanishing gradient ou simplement juste rallonger le temps de la Backpropagation. Dans notre cas, on se limitera à 2 couches cachées avec 100 neurones sur chacune donnant déjà de très bons résultats.

- Fonction d'activation

  La fonction d'activation est un autre élément important à choisir, c'est elle qui va introduire de la non-linéarité dans notre modèle. On retrouve historiquement 3 fonctions d'activations.

- Sigmoïde : $f(x) = \frac{1}{1+e^{-x}}$
  Le sigmoïde était une fonction beaucoup utilisée, mais elle est un peu dépassée notamment puisqu'elle ne converge pas vers l'identité en 0, une propriété qui si elle était vérifiée garantirait un apprentissage rapide avec une initialisation quelconque. On ne va donc pas l'utiliser et on va donc plutôt utiliser :
- ReLU : f(x) = $\begin{cases} 0 \text{ si } x < 0 \\ x \text{ si } x \geq 0 \end{cases}$
  C'est la fonction que l'on va utiliser en majorité pour plusieurs raisons, d'une part, elle possède une dérivée monotone, une propriété qui va avoir tendance à réduire l'overfitting. D'autre part contrairement à d'autres fonctions d'activations, elle va renvoyer moins de valeurs proches de 0 et donc réduire significativement les risques de vanishing Gradient.Cependant, ReLu possède une étendue infinie et donc peut entraîner une instabilité dans l'apprentissage et donc dans certains cas (notamment si améliore l'ordre de plus de 1 ordre) on utilisera plutôt :
- Tangeante Hyperbolique : f(x) = $f(x) = \tanh(x)$

- Algorithme d'optimisation

  On doit aussi choisir l'algorithme d'optimisation de nos poids, tous les algorithmes sont basés sur une descente de gradient, mais il y a des variantes. La variante qui aujourd'hui marche le mieux sur des taches complexes et celle qu'on va donc utiliser est Adam. Nous ne détaillerons pas l'algorithme ici, mais le principe de base de l'algorithme est de calculer pour chaque poids un learning rate adaptatif à partir d'estimateurs des 2 premiers moments du gradient [3].

- Learning rate

  Enfin, le paramètre le plus important à régler est probablement le Learning rate. Le learning rate est le pas d'optimisation de la fonction de perte. Ainsi, il est important d'avoir un pas adapté à la perte, si le pas est trop grand alors on va jamais attendre avec une précision suffisante le minimum globale de la fonction et au contraire si le pas est trop petit alors on peut mettre trop de temps à atteindre le minimum global ou même atteindre un minimum local différent du globale et rester dedans. La fonction de perte étant souvent difficile à étudier analytiquement, on va tester plusieurs valeurs du pas avec un GridSchearch ou un RandomSchearch. Comme le learning rate dans Adam est adapté durant l'apprentissage, ici, on va simplement faire un GridSchearch avec quelques valeurs pour déterminer un bon ordre de grandeur du learning rate.

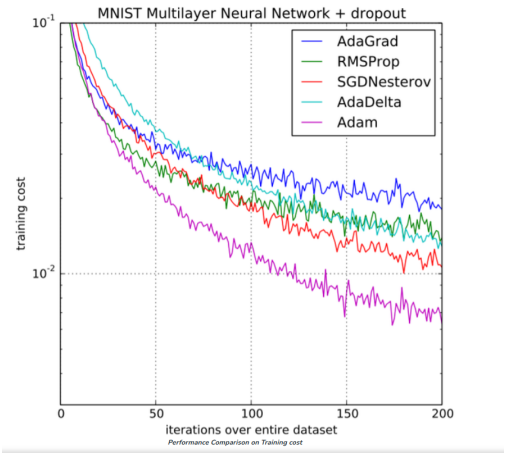

Figure 3: Comparaison de la fonction de perte au cours des epochs pour différents algorithmes classiques d'optimisations sur un problème complexe(reconaissance d'images)

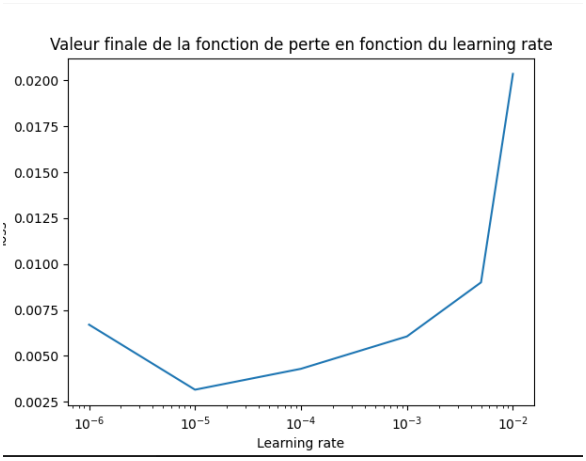

Figure 4: Valeur finale de la fonction de perte en fonction du learning rate de base pour le problème Linéaire avec 50 epochs

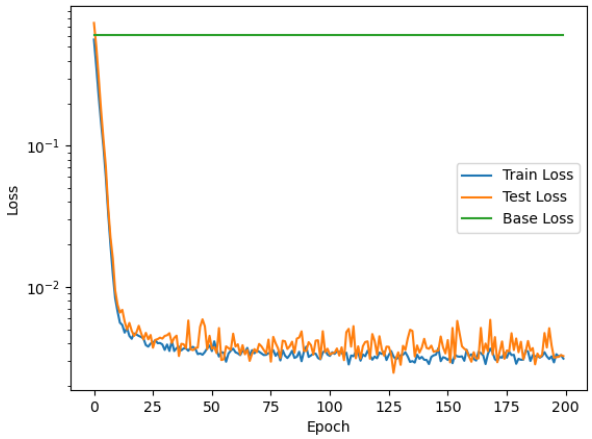

Figure 5: Evolution de la fonction de perte 3.5 sur l'approximation de l'équation modifié Linéaire 2.12 en fonction du nombre d'epochs (Passage sur l'ensemble des données d'entrainement)

On voit bien que le Learning rate doit être dans une plage de valeur pour donner un résultat convenable. Ici en prendras le Learning rate entre 10^{-4} et 10^{-5}.

4 Simulation numérique

4.1 Cas Linéaire

En entrainant notre réseau de neurones sur des valeurs de Y_0 dans l'intervalle [0,2], on peut observer les f_i et σ_i sur cette plage de valeur.

On peut observer dans un premier temps l'évolution de la fonction de perte, on voit qu'avec une bonne valeur de η, on obtient une baisse significative de la différence entre les 2 métriques assez rapidement avant de se stabiliser. Si on compare à la valeur de cette même fonction de perte sans modification. Pour vérifier ça. Mais cette perte ne donne pas toute l'information, pour vérifier ça, on va tracer l'évolution de l'erreur faible en fonction de h. Pour le schéma non modifié, on bien une erreur faible d'ordre 1, alors que pour notre fonction, on est autour de l'ordre faible 2 ou 3, c'est ce qui est attendus. On observe cette erreur pour des pas h relativement grands car on estime l'espérance par un estimateur simple de Monte-Carlo et due au fait qu'on couple cela avec du Machine Learning qui demande déjà beaucoup de ressources computationnelles, on est limité sur le nombre de trajectoires que l'on peut simuler entrainant une assez grande erreur de Monte-Carlo.

On peut aussi observer la différence entre les fonctions calculées par les MLPs

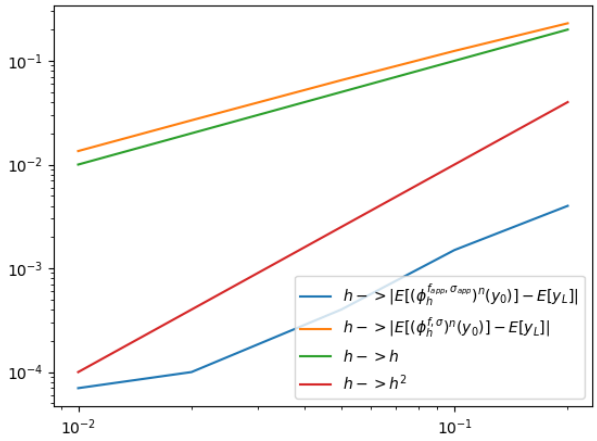

Figure 6: Evolution du l'erreur faible en fonction du pas de la méthode numérique h pour Euler Maruyama et pour Euler Maruyama modifié grâce à notre approximation appliqué à l'équation stochastique Linéaire 2.12

et les valeurs théoriques calculés précédemment. Sur la plage de valeur qui nous a servi pour l'entraînement, on voit bien que nos différents MLPs ont réussi à approximer les valeurs théoriques des f_i et σ_i.

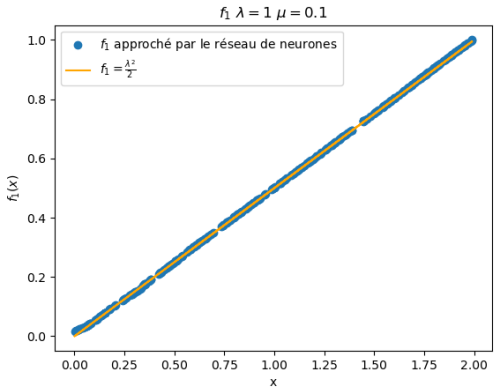

Figure 7: Comparaison entre le premier terme de diffusion f_1 calculé précédement et approximé par notre réseau

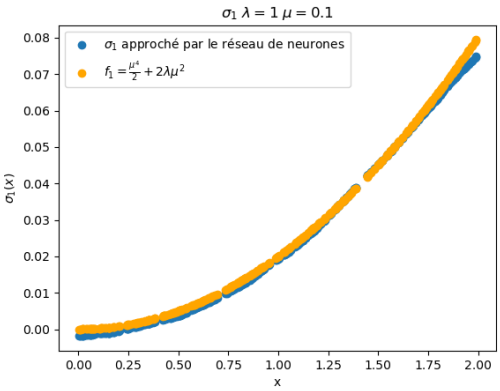

Figure 8: Comparaison entre le premier terme de diffusion σ_1 calculé précédement et approximé par notre réseau

On voit bien que sous une certaine erreur et en dimension 1, les différentes parties de la fonction modifiées peuvent être approximées par un réseau de neurones entraîné pour minimiser notre fonction de perte.

4.2 Pendule Stochastique

Dans un premier temps, on entraîne notre réseau de neurones pour des valeurs dans l'espace $[-\pi, \pi] \times [-1.5, 1.5]$, cependant dans notre cas la complexité de calcul due au mélange Machine Learning et Stochastique peuvent rendre le calcule avec un nombre de points suffisant assez long. Cependant, on sait que la solution exacte vit sur y $H(y) = H(y0)$ afin de réduire notre domaine, on va donc choisir nos points dans un tube autour d'une ellipse approchant la phase réelle. Notre nouveau domaine devient

$$\begin{pmatrix} \frac{\pi}{3}\cos(\theta) + \epsilon_1 \\ \sin(\theta) + \epsilon_2 \end{pmatrix} \theta \in [0, 2\pi], \epsilon_1\epsilon_2 \in [-0.2, 0.2] \tag{4.1}$$

Ainsi, on peut voir qu'avec un nombre de points raisonnable, on couvre bien mieux le domaine.

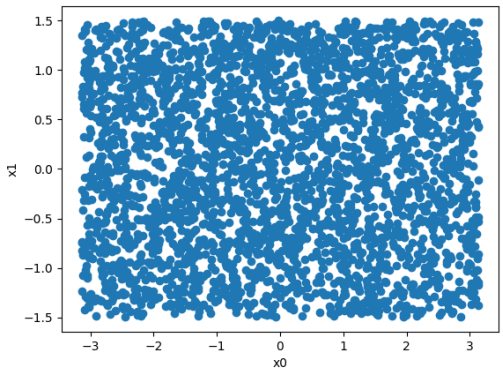

Figure 9: Génération de 1000 points dans le domaine de base $[-\pi, \pi] \times [-1.5, 1.5]$

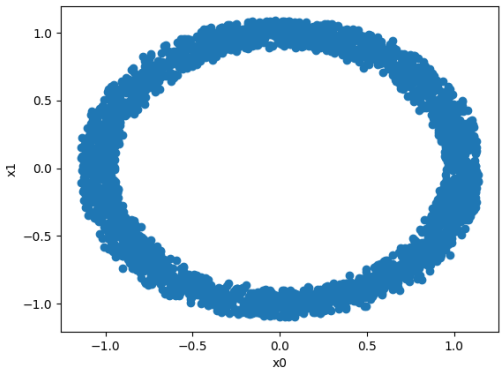

Figure 10: Génération de 1000 points dans le nouveau domaine définit (4.1)

La fonction de perte à une évolution cohérente, elle décroît bien et est réduite de l'ordre de 10^{-1}, on constate que par rapport au problème linéaire la perte met plus de temps avant de converger, on constate aussi qu'il y a plus d'overfiting, un problème qui a été en partie réglé par l'ajout d'un dropout, et aussi par le changement de notre fonction d'activation par ReLu.

De même qu'avec le cas Linéaire, on va observer l'erreur faible en fonction de h pour confirmer l'évolution de la fonction de perte. On prend une plage de pas assez faible [0.1,0.5] d'une part, car on utilise un point fixe qui converge seulement si $h \leq 0.5$ pour appliquer Point Milieu et d'autre part, car l'application du schéma est assez coûteux et donc dans le cadre de nos simulations, on va restreindre le nombre de trajectoires imposant à cause de l'erreur de Monte-Carlo un pas élevée. On voit bien que sur cette plage la constante de l'erreur faible à diminuer, on se

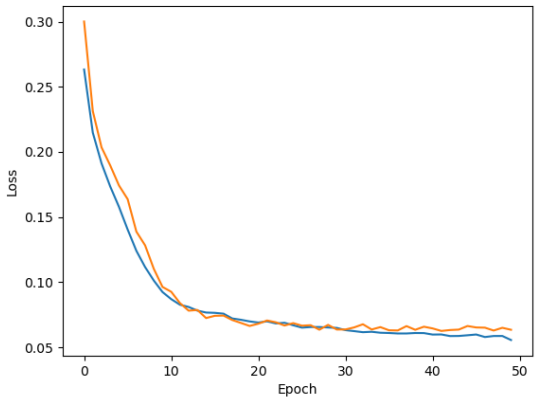

Figure 11: Evolution de la fonction de perte appliqué au pendule stochastique

rapproche bien aussi d'un ordre 2 ce qui est attendu, Point Milieu stochastique étant d'ordre 1 comme EMaruyama, mais avec un coefficient plus faible.

Une propriété importante de Midpoint est que c'est une méthode symplectique, la théorie de l'analyse rétrograde prouve que la méthode améliorée sur base d'une méthode symplectique est aussi symplectique, on aimerais donc que la nouvelle méthode approchée par machine learning conserve cette propriété. On va donc observer l'erreur entre $\tilde{H}(y_t)$ notre nouvel Hamiltonien et $H(y_0)$. On voit que sur des temps faibles, on a bien la conservation de l'Hamiltonien à une erreur similaire à celle produite par la méthode non améliorée. Cependant, sur des temps longs, on n'a pas quelque chose de symplectique. Une solution pour les systèmes hamiltoniens est d'encoder directement la géométrie du problème dans le réseau de neurones en incluant l'Hamiltonien dans la perte.

5 Conclusion

Pendant ce stage, on avait pour but de construire un modèle permettant d'approximer un intégrateur numérique modifié à partir d'un intégrateur numérique de base. On a donc construit un réseau de neurones basé sur plusieurs Multi layer Perceptron en gardant la structure du problème. Après avoir étudié l'influence des différents hyperparamètres et les réglés au mieux sur notre problème. On a utilisé notre réseau sur 2 exemples afin d'une part de comparer notre approximation aux résultats analytiques que l'on a calculée. Et d'autre part afin de voir l'influence de paramètres comme la dimension, l'ordre de troncature de l'équation modifié ou la géométrie du problème de base. On a pu observer que malgré certaines limitations, le modèle donne des résultats cohérents et on peut continuer sur cette base.

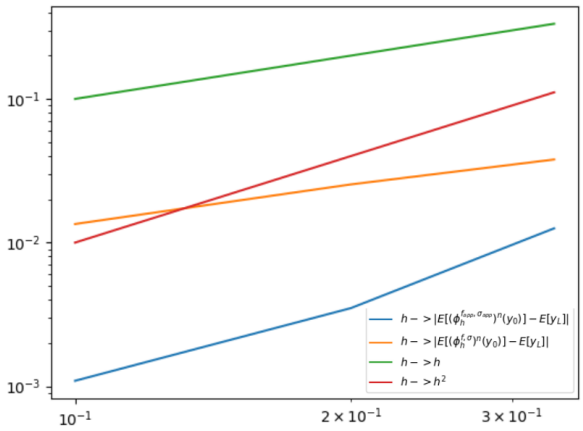

Figure 12: Evolution du l'erreur faible en fonction du pas de la méthode numérique h pour Point Milieu et pour Point Milieu modifié grâce à notre approximation appliqué au pendule stochastique 2.15

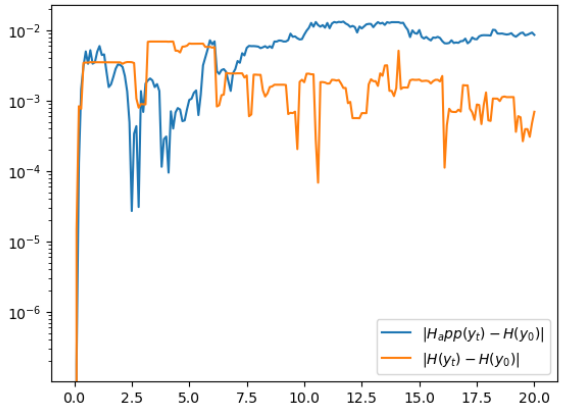

Figure 13: Evolution de l'erreur sur l'hamiltonien en fonction du temps en appliquant Point Milieu et Point Milieu modifié grâce à notre approximation applqué au pendule stochastique 2.15

Cependant, il reste beaucoup de choses à faire. On pourrait d'une part tester notre modèle sur d'autres exemples. On pourrait par exemple étudier sur une EDS avec bruit additif et ainsi se pencher sur la mesure invariante plutôt que l'erreur faible. On pourrait aussi étudier le cas général même si cela ne marche pas totalement théoriquement afin de voir si l'approximation reste correcte. La problématique de temps de calcul direct des coefficients modifiés apparaissant particulièrement à haute dimension, il faudrait étudier pour de plus grandes dimensions et tronquer à des ordres plus grands. Enfin, il faudrait étudier plus en détails la complexité de l'apprentissage afin de l'améliorer. On pourrait aussi utiliser un autre langage différent de python/pytorch afin d'utiliser pleinement la puissance du GPU tout en gérant mieux les ressources allouées, cependant cela nécessiterait beaucoup plus de temps à implémenter.

References

[1] A. Abdulle, D. Cohen, G. Vilmart, and K. C. Zygalakis. High weak order methods for stochastic differential equations based on modified equations. *SIAM J. Sci. Comput.*, 34(3):A1800–A1823, 2012.

[2] M. Bouchereau, P. Chartier, M. Lemou, and F. Méhats. Machine learning methods for autonomous ordinary differential equations. *Submitted*, 2023.

[3] L. B. Kingma. Adam : A method for stochastic optimization.

[4] A. Laurent and G. Vilmart. Exotic aromatic B-series for the study of long time integrators for a class of ergodic SDEs. *Math. Comp.*, 89(321):169–202, 2020.

[5] T. Shardlow. Modified equations for stochastic differential equations. *BIT Numer. Math.*, 46(1):111–125, 2006.