COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Mathématiques et leurs Interactions*

Par
Maxime BOUCHEREAU

Modélisation de phénomènes hautement oscillants par réseaux de neurones

Thèse présentée et soutenue à Rennes, le 17 Octobre 2024
Unité de recherche : UMR CNRS 6625 - Institut de Recherche Mathématique de Rennes (IRMAR)

Rapporteurs avant soutenance :

Bruno DESPRES Professeur, LJLL, Sorbonne Université
Marjolaine PUEL Professeur, Laboratoire AGM, Cergy Paris Université

Composition du Jury :

Président :	Julien MATHIAUD	Professeur, IRMAR, Université de Rennes
Examinateurs :	Philippe CHARTIER	Directeur de Recherche, ININRIA Rennes, en disponiblité chez Ravel Technologies
	Bruno DESPRES	Professeur, LJLL, Sorbonne Université
	Marjolaine PUEL	Professeur, Laboratoire AGM, Cergy Paris Université
	Nicolas SEGUIN	Directeur de Recherche, Antenne INRIA de l'Université de Montpellier
Dir. de thèse :	François CASTELLA	Professeur, IRMAR, Université de Rennes

Remerciements

Ca y est, me voici arrivé au terme de ma thèse. Si j'y suis parvenu, c'est grâce à de nombreuses personnes qui y ont contribué, de loin comme de près, que je tiens ici à remercier.

Tout d'abord, je tiens à remercier François, Philippe, Mohammed et Florian. Travailler avec vous quatre pendant ces trois années a été un immense plaisir, aussi bien du côté des maths que du côté humain. Je garderais en mémoire les réunions Zoom de la semaine ou les passages au bureau pour régler des détails administratifs. J'ai beaucoup appris, et c'est en bonne partie grâce à vous si je suis toujours aussi intéréssé par la recherche. Merci infiniment pour votre disponibilité ainsi que votre bienveillance, qui ont largement contribué à mener à bien ma thèse.

Je remercie également Bruno Després et Marjolaine Puel d'avoir accepté de rapporter ma thèse ainsi que de lire mes travaux. Merci également à Nicolas Seguin et Julien Mathiaud d'avoir accepté de faire partie du jury.

Je tiens à remercier le CHL et la région Bretagne pour leurs financements.

Si je suis parvenu au terme de ma thèse, je le dois beaucoup aux enseignant.e.s ayant contribué à me donner le goût des sciences et surtout des maths. En particulier, je souhaite remercier mon professeur de seconde année de classe préparatoire, Mr. Rolland. Votre enseignement en seconde année PC a été d'une très grande qualité, et m'a servi de modèle à suivre lors de ma mission d'enseignement. Merci beaucoup pour vos précieux conseils que vous m'avez donné depuis sept ans. J'aimerais également remercier l'ensemble des enseignant.e.s que j'ai pu avoir à Rennes, d'une part en m'ayant facilité mon passage de la PC vers la Licence de maths, et d'autre part en apportant du soutien à notre promo durant la délicate période qu'a représenté le covid.

J'aimerais par ailleurs remercier toutes les personnes du laboratoire dont j'ai pu croiser la route. En particulier, je tiens à remercier Marie-Aude, Florian, Annie, Véronique, Olivier, Eric, Sandra. Merci beaucoup pour votre aide dans les tâches administratives, l'informatique, l'enseignement ou encore la réservation de salle.

Je souhaite également remercier les chercheur.euse.s avec lesquel.le.s j'ai eu le plaisir d'échanger durant ma thèse, aussi bien dans l'enseignement que dans la recherche. En particulier, merci à Miguel, Vincent, Erwan, Nicolas, Léo pour les rapides discussions que nous avons pu avoir. Stéphane, j'ai beaucoup apprécié la coopération pour l'UE APA durant mes deux premières années de thèse. Benjamin, j'ai été ravi que l'on puisse travailler ensemble dans le cadre de l'Analyse Numérique et en garderais un excellent souvenir. Mr. Shih, merci infiniment pour les nombreux échanges que nous avons eu pour l'OM3, et merci de nous avoir confié des CM's, j'ai pris beaucoup de plaisir à les dispenser. Adrien, merci pour l'intérêt que tu as porté à mes travaux, j'ai été ravi de faire une excursion dans le monde stochastique, je souhaite que l'on puisse à l'avenir continuer à discuter de nos travaux respectifs.

Pierre N., merci beaucoup pour ton aide dans la programmation informatique, tout particulièrement pour les simulations Python.

J'aimerais aussi remercier les post.doctorant.e.s avec qui j'ai pu échanger au cours de mes trois années de thèse. Tout particulièrement, merci à Hugo E., Antoine, Nathan, Mewen, Marie, Rémi, Nicolas, avec qui j'ai pu échanger lors d'une pause déjeuner, d'un séminaire ou d'une pause gourmande du mardi, ainsi qu'à Emeric, Mégane, Jérémy, Grégoire, Mériadec, Quentin, Paul, Pierre L.B. et Thomas qui ne sont plus à Rennes mais dont j'ai croisé la route durant mes deux premières années de thèse.

François (ou plutôt Francis), j'ai été ravi de t'avoir comme collègue durant ces six années (eh oui, le temps passe vite depuis le M1 !). Je suis convaincu que tu feras un excellent enseignant et te souhaite une très belle continuation.

Hugo M., j'ai beaucoup apprécié d'échanger avec toi sur nos sujets de recherche respectifs. Au plaisir de te revoir pour discuter modélisation d'épidémies ou IA.

Théo, merci pour m'avoir fait découvrir encore un peu plus la théorie du contrôle (qui s'avère être un domaine de recherche sympathique ;)), et merci infiniment pour ton implication dans l'organisation du séminaire Landau, je pense que nous te devons beaucoup. Bon courage à toi pour la fin de ta thèse.

Guillaume, merci pour ta bonne humeur et ta gentillesse, je garderais en mémoire nos discussions variées à parler d'Histoire, de maths bien sûr (curieusement, Navier-Stokes et Machine Learning étaient souvent au menu) ou encore de conduite. Je te souhaite bon courage pour ta troisième année de thèse.

Mouhamad, j'ai beaucoup apprécié de t'avoir comme co-bureau durant ma deuxième année de thèse, et pris énormément de plaisir à échanger sur nos sujets de recherche respectifs, probalement par le fait que nous sommes tous les deux très attachés à l'analyse numérique. Au plaisir de se revoir (en Bretagne ou à Lyon).

Mahieddine, comment te remercier ? Tu as été un colègue formidable durant ces quatre années qu'ont été le Master 2 et la thèse. J'ai adoré discuter maths et autres domaines lors d'une pause déjeuner, après une surveillance d'examen ou lors d'une préparation de TD en OM3. J'ai été ravi de t'avoir fait découvrir Saint-Malo (qui est au passage une magnifique ville). Je te souhaite une très bonne continuation comme enseignant à Troyes.

Bien entendu, une thèse réussie n'est pas uniquement le fruit de personnes en lien avec les maths, beaucoup de monde extérieur a contribué à ce que je la mène à son terme.

Je tiens à remercier les ami.e.s qui m'ont soutenu et avec qui j'ai partagé de bons moments durant ces trois années.

Charles, en plus d'avoir été un très bon camarade durant mes deux premières années rennaises, tu m'as apporté beaucoup de beinveillance, de soutien et de conseils durant ma thèse. Merci pour tout, je te souhaite le meilleur pour la suite.

Je tiens aussi à remercier les gens du FJT. Tout particulièrement, merci à Théo & Théo, Amélie, Eloïse et Daun. Merci pour les soirées au FJT ou au Black Bear, les avantures au Blizz d'été (à tomber par terre :D) ou encore les petites virées dans les Côtes d'Armor. Merci pour tous ces moments partagés.

Pauline, merci d'être ma meilleure amie depuis onze ans, d'avoir été à l'écoute, pour tous les moments partagés (y compris les bavardages en Arts plastiques :)). Brian, je suis ravi que nous nous sommes rencontrés au début de ma première année. Merci à vous deux pour votre gentillesse, votre bonne humeur et votre soutien.

Merci également à Dominique et Catherine, Sylvie et Jean-Yves ainsi que Jocelyne et Raymond, pour l'intérêt que vous avez manifesté pour ma thèse. J'ai pris beaucoup de plaisir à vous décrire mes travaux avec mes mots.

Il va également de soi que je dois en grande partie la réussite de ma thèse à ma famille.

J'ai évidemment une pensée nostalgique pour les personnes nous ayant quittés et qui auraient aimé me voir terminer ma thèse.

Je tiens à remercier mon parrain Dominique et ma tante Sylvie pour votre gentillesse ainsi que votre intérêt pour ma thèse. Merci pour tous les bons moments passés en votre compagnie.

Un grand merci à ma marraine Christiane. Merci pour ta gentillesse et ta bienveillance, ainsi que ton soutien et tes encouragements dans les moments les plus délicats. Merci aussi pour tous tes petits conseils qui m'ont permis de découvrir la Bretagne (une si belle région) durant mes trois années de thèse.

Juju et Jo, merci à vous deux pour les bons moments passés à discuter (même si je fais des maths compliquées ;)) et rigoler. Merci à toi Juju pour tous les bon repas (il faut le reconnaître, tu surclasses ton petit frère en cuisine ;)). Jo, merci pour ta bonne humeur à faire rire tout le monde. Merci à ma filleule Chloé, je suis très fier d'être ton parrain depuis deux ans. Parrain te fait un p'tit coucou.

Enfin, je tiens à remercier mes parents pour tout ce qu'ils m'ont apporté jusqu'à présent. Merci Papa, merci Maman. Si aujourd'hui je suis arrivé au terme de ma thèse, c'est en très grande partie grâce à vous. Merci pour votre soutien et votre bienveillance, merci pour vos encouragements et pour tous ces bons moments passés ensemble. Du fond du coeur, merci.

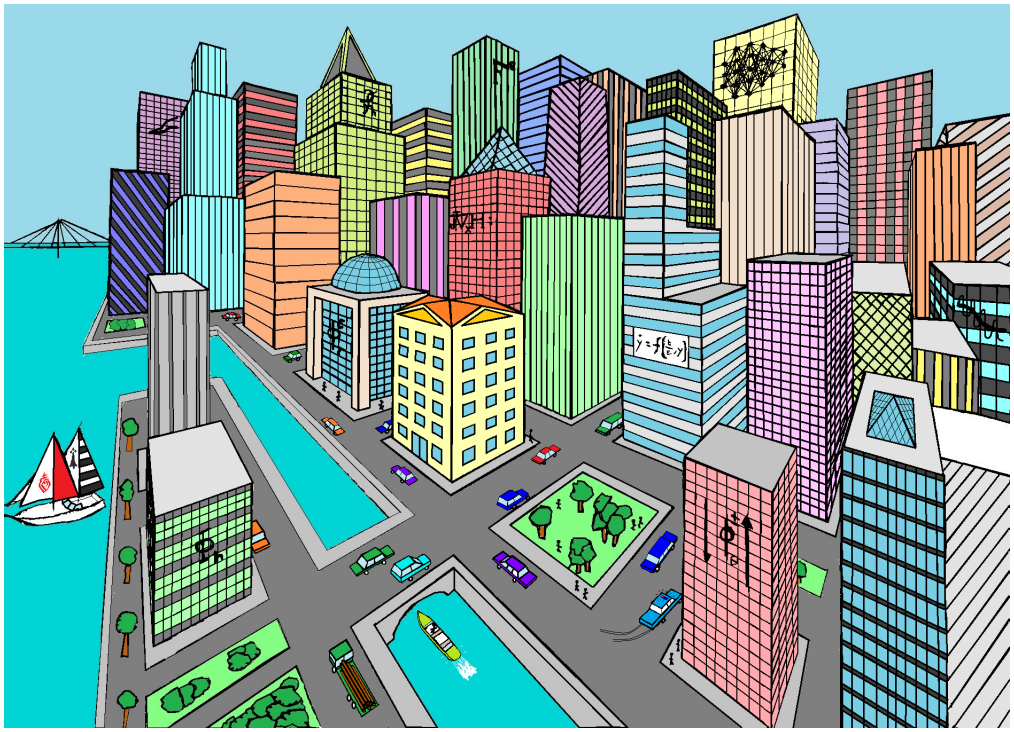

Figure 1 – Quelques éléments de ma thèse sont partis se promener en ville, amusez-vous à les retrouver :)

Table of Contents

Introduction **15**

1 Motivation . . . 15

2 Boîte à outils . . . 16

2.1 Intégration numérique des équations différentielles autonomes 16

2.1.1 Equation modifiée . . . 16

2.1.2 Equations hamiltoniennes . . . 19

2.1.2.1 Généralités . . . 19

2.1.2.2 Intégration numérique des équations hamiltoniennes. . . . 20

2.1.3 Lemme de Grönwall discret . . . 21

2.2 Equations différentielles fortement oscillantes . . . 22

2.2.1 Précédents travaux . . . 23

2.2.1.1 Théorie . . . 23

2.2.1.2 Méthodes numériques . . . 24

2.2.2 Définitions et premières propriétés . . . 26

2.2.2.1 Champ moyen . . . 26

2.2.2.2 Décomposition en deux dynamiques . . . 27

2.2.2.3 Champ moyenné et changement de variable 28

2.2.2.4 Méthode d'approximation du champ moyenné et du changement de variable . . . 28

2.2.3 Décomposition en deux dynamiques avec reste exponentiel . . . 29

2.2.4 Cas autonome fortement oscillant . . . 29

2.2.5 Méthodes UA . . . 30

2.2.5.1 Décomposition Micro-macro . . . 31

2.2.5.2 Schéma Micro-macro integral . . . 32

2.2.5.3 Méthode Pullback . . . 32

2.3 Réseaux de neurones . . . 33

2.3.1 Réseaux de neurones et équations différentielles: précédents travaux . . . 33
2.3.1.1 Réseaux de neurones profonds . . . 33
2.3.1.2 PINN's . . . 35
2.3.2 Définitions et propriétés d'approximation . . . 36
2.3.3 Approximation d'une fonction avec un réseau de neurones . . . 37
2.3.3.1 Collecte de données et entraînement . . . 37
2.3.3.2 Algorithme de descente de gradient . . . 39
2.3.4 Ajout de structure sur les réseaux de neurones . . . 40
2.3.4.1 Perturbation d'un champ connu . . . 40
2.3.4.2 Structure d'auto-encodeur . . . 41
2.3.4.3 Résolution d'équations différentielles par les PINN's . . . 41
3 Plan du manuscrit . . . 43
3.1 Chapitre 1: Machine Learning et Equations autonomes . . . 43
3.2 Chapitre 2: Amélioration d'un schéma intégral d'ordre 1 via des réseaux de Neurones . . . 46
3.3 Chapitre 3: Deep Learning pour les Equations fortement oscillantes . . . 49
3.3.1 Décomposition en deux dynamiques . . . 49
3.3.2 Ajout d'une correction Micro-Macro . . . 53
3.3.3 Cas autonome fortement oscillant . . . 54
3.4 Chapitre 4: PINN's pour le cas fortement oscillant . . . 57
4 Perspectives de recherche . . . 58
4.1 Propriétés géométriques . . . 58
4.2 Dépendance en ε . . . 60
5 Articles de recherche . . . 61

Thesis work 64

1 Machine Learning methods for Autonomous differential equations 65
1.1 Introduction . . . 65
1.1.1 Scope of the chapter . . . 67

1.1.2 Related work 68
1.2 Improving the accuracy of numerical methods with machine learning 69
1.2.1 General strategy 69
1.2.1.1 Modified field 70
1.2.1.2 Machine learning methods 70
1.2.2 Error analysis 72
1.2.2.1 General case 72
1.2.2.2 Explicit Runge-Kutta method 74
1.2.3 An alternative method for parallel training 75
1.3 Numerical experiments 75
1.3.1 Approximation of the modified field 77
1.3.2 Loss decay and Integration of ODE's 79
1.3.3 Computational times for explicit methods 85
1.3.4 Evaluation of an alternative method 87
1.4 Conclusions 88
1.A Appendix: Proof of Theorem 24 88
1.B Appendix: Proof of Theorem 25 90
1.C Appendix: Choice of the parameters 93
1.C.1 Link between learning error and parameters 93
1.C.2 Parameters selected in our numerical experiments 94
1.C.2.1 Nonlinear Pendulum - Forward Euler 94
1.C.2.2 Rigid body system - Forward Euler 95
1.C.2.3 Nonlinear Pendulum - Runge-Kutta 2 95
1.C.2.4 Nonlinear Pendulum - midpoint 96
1.C.3 Nonlinear Pendulum - Comparison between traditional and an alternative method 96

2 First order Integral scheme performing with Neural Network 98
2.1 Introduction 98
2.2 Improving the accuracy of a numerical scheme with machine learning 99
2.2.1 General strategy 99
2.2.1.1 Forward Euler Integral scheme 99
2.2.1.2 The help of backward error analysis 100
2.2.1.3 Machine learning method 100

2.2.2 Error analysis . . . 103
2.3 Numerical experiment . . . 103
2.3.1 Loss decay and Integration of ODE's . . . 104
2.3.2 Computational times . . . 109
2.4 Conclusions . . . 111
2.A Appendix: Proof of Proposition 26 . . . 111
2.B Appendix: Proof of Theorem 27 . . . 113
2.C Appendix: Implementations . . . 115
2.C.1 Numerical integration . . . 115
2.C.2 Choice of the parameters for numerical experiments . . . 116

3 Deep Learning for Highly oscillatory differential equations 117
3.1 Introduction . . . 117
3.1.1 Scope of the paper . . . 118
3.1.2 Related work . . . 119
3.2 Approximate solutions of highly oscillatory differential equations with machine learning . . . 120
3.2.1 General strategy . . . 120
3.2.1.1 Slow-fast decomposition . . . 121
3.2.1.2 Micro-Macro decomposition . . . 122
3.2.1.3 Machine learning method . . . 122
3.2.1.4 An alternative method for autonomous systems . . . 126
3.2.2 Error analysis . . . 128
3.2.2.1 Slow-fast decomposition . . . 129
3.2.2.2 Micro-Macro correction . . . 130
3.2.2.3 Alternative method . . . 132
3.3 Numerical experiments . . . 132
3.3.1 Approximation of the averaged field and high oscillation generator . . . 134
3.3.2 Loss decay and integration of ODE's . . . 135
3.3.2.1 Inverted Pendulum - forward Euler method . . . 137
3.3.2.2 Inverted Pendulum - midpoint method . . . 141
3.3.2.3 Van der Pol oscillator - forward Euler method . . . 145
3.3.3 Error curves w.r.t. step size . . . 152

3.3.4 Uniform accuracy test . . . 154
3.3.5 Evaluation of alternative method . . . 156
3.4 Conclusions . . . 163
3.A Appendix: Proof of Theorem 28 . . . 163
3.B Appendix: Proof of Theorem 29 . . . 165
3.C Appendix: Proof of Theorem 30 . . . 168
3.D Appendix: Implementation of implicit methods . . . 169
3.E Appendix: Computation of learning errors . . . 170
3.E.1 Space and time discretizations . . . 170
3.E.2 Intergals and derivatives representation . . . 170
3.F Appendix: Influence of learning error . . . 171
3.G Appendix: Choice of the parameters . . . 172
3.G.1 Inverted Pendulum - Forward Euler method . . . 172
3.G.2 Inverted Pendululm - midpoint method . . . 172
3.G.3 Van der Pol oscillator - Forward Euler method . . . 173
3.G.4 Van der Pol oscillator: Comparison between classical and alternative method . . . 173

4 PINN's for highly oscillatory equations 174
4.1 Introduction . . . 174
4.2 Solving highly oscillatory ODE's with PINN's . . . 174
4.2.1 General strategy . . . 174
4.2.2 Machine Learning method . . . 175
4.2.3 Neural Network structure . . . 176
4.2.4 Error analysis . . . 177
4.3 Numerical experiments . . . 177
4.4 Conclusions . . . 194
4.A Appendix: Proof of Theorem 31 . . . 194
4.B Appendix: Implementations . . . 195
4.B.1 Finite difference . . . 195
4.B.2 Choice of the parameters for numerical experiments . . . 196
4.B.2.1 Hénon-Heiles system . . . 196
4.B.2.2 Van der Pol oscillator . . . 196

Bibliography 197

INTRODUCTION

1 Motivation

Les travaux réalisés durant cette thèse portent sur l'application de méthodes de Machine Learning à l'étude d'équations différentielles fortement oscillantes de la forme

$$\dot{y^\varepsilon}(t) = f\left(\frac{t}{\varepsilon}, y^\varepsilon(t)\right), \tag{1}$$

où $f : \mathbb{R} \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ est une fonction régulière, $2\pi-$périodique en sa première variable et $\varepsilon \in]0, 1]$. Lorsque ε est très petit, le système est en régime fortement oscillant.

Dans le cas d'équations différentielles non oscillantes, de nombreuses méthodes numériques existent et en permettent une résolution efficace. Cependant, ces méthodes sont inadaptées pour les équations fortement oscillantes. En effet, les estimations d'erreurs liées à ces méthodes numériques font intervenir les dérivées du champ de vecteurs associé à l'équation différentielle, dérivées qui, dans le cas fortement oscillant, ne sont pas bornées lorsque ε devient petit, ayant pour effet de dégrader l'erreur numérique. Pour cette raison, des méthodes numériques spécifiquement adaptées à ce type d'équation ont été développées. Bien que très performantes, ces méthodes nécessitent des calculs préparatoires (potentiellement complexes) afin de pouvoir être mises en oeuvre.

Les travaux réalisés durant cette thèse proposent d'utiliser des techniques de Machine Learning afin d'approcher la solution d'équations différentielles fortement oscillantes de manière précise tout en évitant le recours aux calculs préparatoires complexes. Dans certains cas, une approximation uniforme par rapport au paramètre ε peut être obtenue.

2 Boîte à outils

2.1 Intégration numérique des équations différentielles autonomes

Les équations différentielles autonomes constituent une classe importante (et l'une des plus simples) d'équations différentielles. Elles sont utilisées pour modéliser différents phénomènes en physique, en biologie, etc... Cependant, en dépit de leur simplicité, elles sont rarement résolubles de manière analytique. Pour cette raison, de nombreuses méthodes numériques ont été développées afin d'approcher les solutions exactes de ces équations via des approximations numériques [48].

Parmi ces méthodes, les méthodes numériques à un pas telles que les méthodes de Runge-Kutta en constituent une classe importante.

Dans toute cette partie, nous considèrerons des systèmes autonomes de la forme

$$\left\{ \begin{array}{rcl} \dot{y}(t) & = & f(y(t)) \\ y(0) & = & y_0 \in \mathbb{R}^d \end{array} \right. \tag{2}$$

où $f : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ est supposée régulière, vérifiant les conditions du théorème de Cauchy-Lipshitz.

2.1.1 Equation modifiée

Afin d'avoir une étude plus aisée des méthodes numériques, on peut utiliser l'analyse rétrograde [50], qui consiste, pour une méthode numérique donnée, à faire une perturbation du champ de vecteurs associé à une équation différentielle autonome de la forme (2) puis à utiliser la méthode avec ce nouveau champ de vecteurs afin d'approcher plus précisément le flot exact. Plus précisément, on a:

Définition 1. ***Champ modifié****. Soient Φ_h^f une méthode numérique à un pas, appliquée avec le champ f et φ_h^f le flot exact associé à f. On définit le champ modifié de f associé à la méthode Φ_h, par le champ de vecteurs, noté $\tilde{f}_h$, tel que, pour tout $y \in \mathbb{R}^d$:*

$$\Phi_h^{\tilde{f}_h}(y) = \varphi_h^f(y) \tag{3}$$

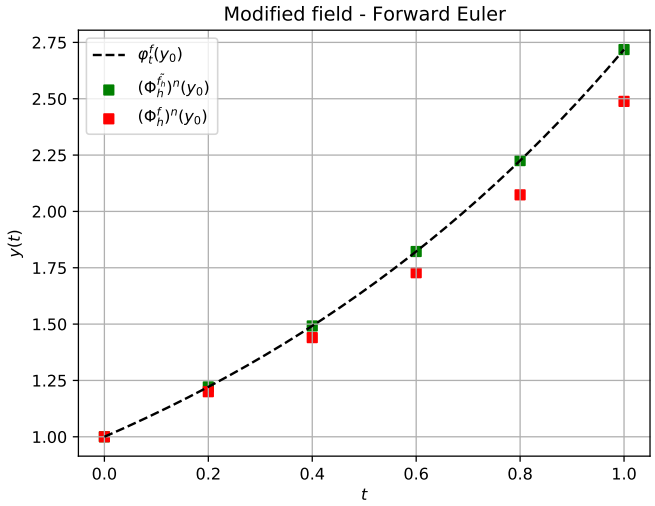

Figure 2 – Exemple d'équation modifiée avec la méthode d'Euler explicite pour l'équation $\dot{y} = y$. Le flot exact est tracé en trait pointillé noir, le flot numérique avec $f : y \mapsto y$ est tracé en points rouge, et le flot numérique avec le champ modifié (correspondant au flot exact) est tracé en points vert. Le pas de temps choisi est $h = 0.2$.

Définition 2. ***Ordre d'une méthode numérique***. *Une méthode numérique Φ_h^f est dite d'ordre $p \in \mathbb{N}^*$ si, pour N itérations du schéma, on a*

$$\underset{0 \leqslant n \leqslant N}{Max} \left| \left(\Phi_h^f\right)^n (y_0) - \varphi_{nh}^f(y_0) \right| \leqslant Ch^p \tag{4}$$

où la constante $C > 0$ ne dépend que de Nh (temps d'intégration).

Proposition 3. ***Structure du champ modifié***. *Le champ modifié $\tilde{f}_h$ par rapport à une méthode numérique d'ordre p a la structure suivante: pour $\Omega \subset \mathbb{R}^d$, compact, pour $q \in \mathbb{N}^*$, pour tout $y \in \Omega$ et pour $h > 0$ suffisamment petit,*

$$\tilde{f}_h(y) = f(y) + h^p f_1(y) + \cdots + h^{p+q-2} f_{q-1}(y) + h^{p+q-1} R(y,h) \tag{5}$$

où $f_1, \cdots, f_{q-1} : \Omega \longrightarrow \mathbb{R}^d$ et $R : \Omega \times \mathbb{R} \longrightarrow \mathbb{R}^d$ sont des fonctions régulières.

Remarque. *Dans le cas de la méthode du point milieu (implicite), les termes devant les puissances impaires de h sont nuls [20, 50].*

A l'exception du cas linéaire, le champ modifié est en général une série formelle (autrement dit cette série ne converge pas forcément), nous incitant ainsi à utiliser une troncature de ce dernier afin d'augmenter l'ordre d'une méthode numérique:

Proposition 4. ***Champ modifié tronqué***. *Considérons le champ modifié tronqué* $\tilde{f}_h^{[q]}$ *:*

$$\tilde{f}_h^{[q]}(y) = f(y) + h^p f_1(y) + \cdots + h^{p+q-2} f_{q-1}(y). \tag{6}$$

Alors nous avons

$$\underset{0\leqslant n\leqslant N}{Max} \left| \left(\Phi_h^{\tilde{f}_h^{[q]}}\right)^n (y_0) - \varphi_{nh}^f(y_0) \right| \leqslant C' h^{p+q-1} \tag{7}$$

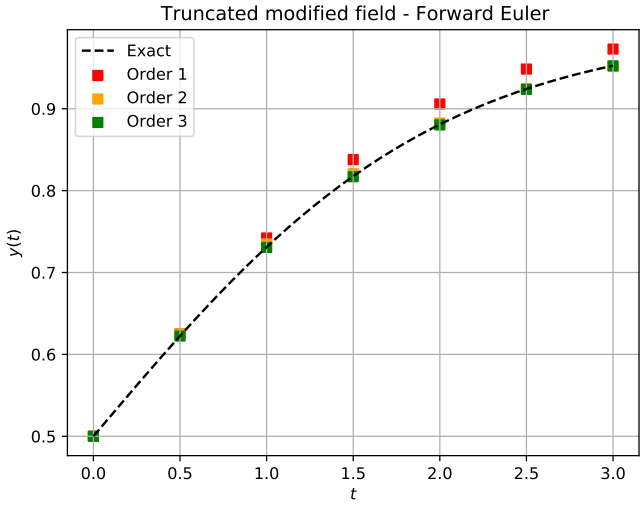

Figure 3 – Exemple d'intégration numérique avec un champ modifié tronqué pour la méthode d'Euler explicite sur l'équation $\dot{y} = y(1-y) =: f(y)$. Le flot exact est tracé en pointillé noir, le flot numérique avec $\tilde{f}_h^{[1]} = f$ est tracé en points rouge, le flot numérique avec $\tilde{f}_h^{[2]}$ est en points orange et le flot numérique avec $\tilde{f}_h^{[3]} = f$ est en points vert. Le pas de temps choisi est $h = 0.5$.

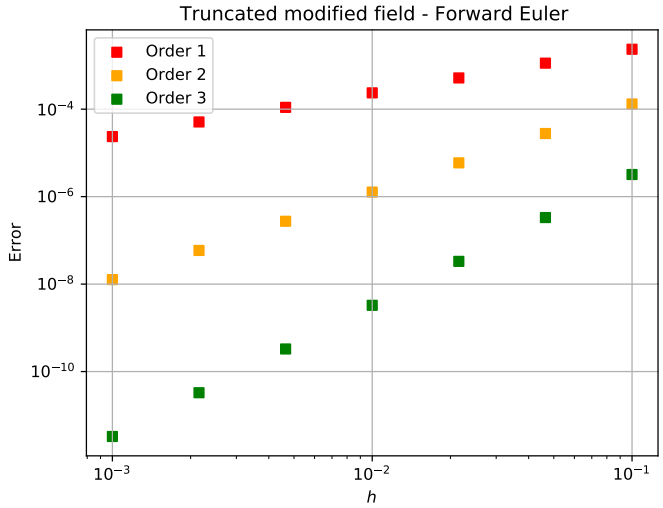

Figure 4 – Exemple de montée en ordre de convergence en utilisant l'intégration numérique avec le champ modifié tronqué pour la méthode d'Euler explicite sur l'équation $\dot{y} = y(1-y) =: f(y)$. L'erreur numérique pour l'intégration avec $\tilde{f}_h^{[1]} = f$ est tracée en points rouge, celle pour l'intégration avec $\tilde{f}_h^{[2]}$ est tracée en points orange et celle pour l'intégration avec $\tilde{f}_h^{[3]}$ est tracée en points vert. On a bien une montée en ordre.

2.1.2 Equations hamiltoniennes

Dans certains cas, les solutions d'EDO peuvent préserver une quantité définie (par exemple une énergie). Pour cette raison, les équations hamiltoniennes constituent une importante classe d'EDO, en lien avec la physique.

2.1.2.1 Généralités

Définition 5. ***champ de vecteurs hamiltonien***. *Notons $J \in \mathcal{M}_{2d}(\mathbb{R})$ la matrice symplectique cannonique*

$$J = \begin{bmatrix} 0 & I_d \\ -I_d & 0 \end{bmatrix}. \tag{8}$$

Un champ de vecteurs $f : \mathbb{R}^{2d} \longrightarrow \mathbb{R}^{2d}$ est dit hamiltonien s'il existe une fonction régulière $H : \mathbb{R}^{2d} \longrightarrow \mathbb{R}$ appelée ***Hamiltonien*** *telle que $f = J^{-1}\nabla H$.*

Maintenant, dans cette section, nous considèrerons des équations de la forme

$$\begin{cases} \dot{y}(t) &= J^{-1}\nabla H(y(t)) \\ y(0) &= y_0 \in \mathbb{R}^{2d} \end{cases} \tag{9}$$

appelées *équations Hamiltoniennes.*

Proposition 6. ***Préservation de l'Hamiltonien****. Le flot associé à une équation de la forme* (9) *préserve l'Hamiltonien. En d'autres termes, si* $f = J^{-1}\nabla H$*, alors pour tout* $t \geqslant 0$*,*

$$H\left(\varphi_t^f(y_0)\right) = H(y_0) \tag{10}$$

2.1.2.2 Intégration numérique des équations hamiltoniennes. Comme les équations autonomes "classiques", les équations hamiltoniennes, malgré leur simplicité et leur structure, sont souvent impossibles à résoudre de manière analytique. Ainsi, des méthodes numériques peuvent être employées. Cependant, certaines méthodes ne préservent pas l'Hamiltonien. En d'autres termes, l'Hamiltonien n'est pas conservé par le flot numérique, et ce, bien qu'il soit constant sur le flot exact. Pour cette raison, nous avons besoin d'une classe de méthodes préservant l'Hamiltonien, appelé *méthodes symplectiques* [50].

Définition 7. ***Application symplectique****. Soit* $U \subset \mathbb{R}^{2d}$ *un ouvert. Une application régulière* $f : U \longrightarrow \mathbb{R}^{2d}$ *est dite symplectique si pour tout* $y \in U$*:*

$$\mathrm{d}f(y)^T \cdot J \cdot \mathrm{d}f(y) = J \tag{11}$$

où $\mathrm{d}f : U \longrightarrow \mathcal{M}_{2d}(\mathbb{R})$ *est la matrice jacobienne de* f*.*

Définition 8. ***Méthode symplectique****. Une méthode numérique à un pas* Φ_h^f *par rapport à* f *est dite symplectique si, pour tout* $h > 0$*, l'application* $y \longmapsto \Phi_h^f(y)$ *est symplectique.*

Proposition 9. ***Champ modifié et structure hamiltonienne****. Si* f *est un champ de vecteurs hamiltonien, alors le champ modifié par rapport à une méthode numérique symplectique d'ordre p possède également une structure hamiltonienne et, pour tout* $y \in \Omega \subset \mathbb{R}^{2d}$ *compact:*

$$\begin{aligned} \tilde{f}_h(y) &= J^{-1}\nabla H(y) + h^p J^{-1}\nabla H_1(y) + \cdots + h^{p+q-2} J^{-1}\nabla H_{q-1}(y) \\ &+ h^{p+q-1} J^{-1}\nabla H_R(y,h) \end{aligned} \tag{12}$$

où $H_1, \cdots, H_{q-1} : \Omega \longrightarrow \mathbb{R}$ *et* $H_R : \Omega \times \mathbb{R} \longrightarrow \mathbb{R}$ *sont des fonctions régulières.*

Proposition 10. *Considérons une méthode numérique symplectique, notée* Φ_h^f*. Alors le flot numérique associé à cette méthode préserve l'Hamiltonien modifié.*

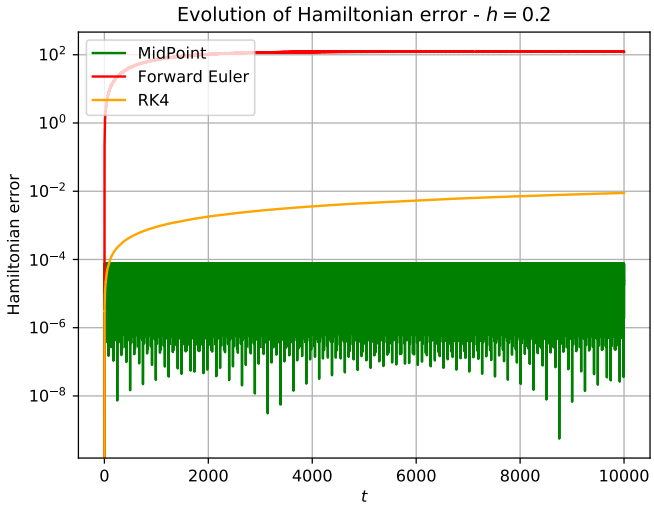

Figure 5 – Exemple de conservation d'Hamiltonien (bien qu'il n'y ait pas conservation au sens strict, l'Hamiltonien reste borné) par la méthode du point milieu implicite (qui est une méthode symplectique) face à la non-conservation avec une méthode non-symplectique (Euler explicite et Runge-Kutta 4) sur l'équation du Pendule simple non linéaire, dont l'Hamiltonien est donné par $H(y) = \frac{1}{2}y_2^2 + 1 - \cos(y_1)$.

2.1.3 Lemme de Grönwall discret

Afin de prouver l'ordre des méthodes numériques et d'obtenir des estimations d'erreur entre le flot numérique et le flot exact dans le cas d'une équation différentielle (autonome en particulier), nous devons:

- Faire des estimations sur **l'erreur de consistance**, en étudiant la quantité

$$\varepsilon_n := y(t_{n+1}) - \Phi_h^f(y(t_n)) \tag{13}$$

où $t_n = nh$, $h = \frac{T}{N}$, N est le nombre total d'itérations.

- Lorsque l'on a des bornes sur ε_n, on peut étudier **l'erreur globale** donnée par

$$e_n := y(t_n) - \left(\Phi_h^f\right)^n (y_0). \tag{14}$$

Lorsque l'on étudie cette quantité, on obtient souvent une estimation de cette forme:

$$|e_{n+1}| \leqslant (1 + Lh)|e_n| + |\varepsilon_n| \tag{15}$$

On effectue alors des estimations supplémentaires en utilisant le lemme de Grönwall discret [54].

Proposition 11. ***Lemme de Grönwall discret****. Soient $(r_n)_{n\in\mathbb{N}}$, $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ trois suites à termes positifs. telles que, pour tout $n \in \mathbb{N}$,*

$$r_{n+1} \leqslant u_{n+1} + \sum_{k=0}^{n} v_k r_k. \tag{16}$$

Alors on a, pour tout $n \in \mathbb{N}$,

$$r_n \leqslant u_n + \sum_{k=0}^{n-1} u_k v_k \exp\left(\sum_{j=k+1}^{n-1} v_j\right) \tag{17}$$

Cependant, la plupart des estimations rencontrées dans le cas des EDO sont de la forme (15) et requièrent une forme plus simple du lemme de Grönwall discret.

Proposition 12. ***Lemme de Grönwall discret - forme simplifiée****. Soient $(r_n)_{n\in\mathbb{N}}$ et $(u_n)_{n\in\mathbb{N}}$ deux suites à termes positifs telles que, pour tout $n \in \mathbb{N}$,*

$$r_{n+1} \leqslant (1 + Lh)r_n + u_n, \tag{18}$$

alors on a

$$r_n \leqslant e^{nLh} r_0 + \sum_{k=0}^{n-1} e^{L(n-k-1)h} u_k \tag{19}$$

2.2 Equations différentielles fortement oscillantes

Les équations différentielles fortement oscillantes constituent un cas particulier du problème multi-échelle. Elles sont largement utilisées pour modéliser des phéno-

mènes physiques ou biologiques présentant des oscillations à différentes échelles. Par exemple, des processus biologiques tels que la démographie d'une population (à longue échelle) influencée par les saisons (à courte échelle), ou des phénomènes physiques tels que la trajectoire d'un pendule (à longue échelle) avec une oscillation forcée (à courte échelle).

Dans toute la section, nous considérons des équations différentielles de la forme

$$\begin{cases} \dot{y^\varepsilon}(t) &= f\left(\frac{t}{\varepsilon}, y^\varepsilon(t)\right) \\ y^\varepsilon(0) &\in \mathbb{R}^d \end{cases} \tag{20}$$

où la fonction $f : \mathbb{R} \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ est supposée être régulière, vérifie les conditions du théorème de Cauchy-Lipschitz et est 2π-périodique par rapport à sa première variable. De plus, le paramètre $\varepsilon \in]0, 1]$ décrit la vitesse des oscillations. En particulier, lorsque ε est petit, l'équation présente des oscillations élevées.

2.2.1 Précédents travaux

Constituant une classe de problèmes multi-échelles, les équations différentielles fortement oscillantes nécessitent des outils spécifiques afin d'être étudiées, tant d'un point de vue analytique qu'au niveau de l'approximation numérique de solutions.

2.2.1.1 Théorie La structure multi-échelle des équations fortement oscillantes offre différentes stratégies exploitant de manière variée cette structure afin d'en étudier le propriétés [97].

Une première stratégie d'étude des équations différentielles fortement oscillantes passe par la méthode de moyennisation de Krylov–Bogoliubov [9], permettant ainsi d'étudier une équation différentielle linéaire du second ordre avec un reste non-linéaire présentant de fortes oscillations. Cette méthode exploite largement l'étude de la décomposition de solutions sous forme d'une série de fonctions.

L'étude analytique de solutions d'équations différentielles fortement oscillantes pouvant exploiter la structure multi-échelle de ces solutions, il est possible d'utiliser la décomposition *slow-fast*. Cette décomposition consiste à formuler la structure des

solutions en deux dynamiques différentes: une dynamique lente, qui utilise la notion de champ moyenné, et une dynamique rapide générant les fortes oscillations [25]. Cette décomposition, calculable à l'aide de séries formelles, diffère de la vraie solution d'un reste exponentiel [24]. C'est la théorie de la moyennisation, qui se décline en deux formes: stroboscopique (afin de faciliter les calculs nécessaires pour trouver la décomposition) ou bien standard (adaptée à l'étude de propriétés géométriques).

Une approche alternative à cette décomposition consiste à étudier la solution de ce type d'équations différentielles sous forme d'un polynôme dont l'indéterminée est le paramètre de fortes oscillations, fournissant par ailleurs des formules ainsi qu'une méthode de moyennisation (et donc d'approximation de solution) à tout ordre [87].

Bien qu'une seule fréquence soit souvent présente dans le cadre d'équations fortmement oscillantes, certains problèmes présentant plusieurs fréquences liées aux oscillations peuvent également être étudiés [26].

Parmi ces équations différentielles, il en existe certaines présentant des propriétés géométriques. L'exemple de base concerne les équations hamiltoniennes, où une énergie (appelée ici Hamiltonien) est préservée. L'étude de solutions de telles équations repose sur les méthodes précédentes, tout en tenant compte du caractère hamiltonien ou symplectique des champs de vecteurs et applications utilisés [16, 25].

Bien que les équations fortement oscillantes concernent majoritairement les équations différentielles, il est également possible d'étudier des équations aux dérivées partielles dont un terme génére de fortes oscillations [19].

2.2.1.2 Méthodes numériques Afin d'approcher numériquement les solutions d'équations différentielles fortement oscillantes, différentes stratégies ont été étudiées. Ces différentes méthodes proposées constituent pour la plupart un pendant numérique des méthodes développées afin d'étudier analytiquement les équations différentielles fortement oscillantes.

Une approche immédiate consiste à utiliser la moyennisation ainsi que la décom-

position *slow-fast* afin de créer des méthodes numériques performantes. L'obtention de cette décomposition passe tout d'abord par un calcul précis et efficace des champs de vecteurs nécessaires, en utilisant des séries formelles [25] ou bien des formules données par récurrence faisant appel à des calculs de dérivées [21] ou bien des différences divisées [22]. Notons que le calcul de dérivées peut être fait numériquement [13].

Certaines de ces méthodes numériques conduisent à des estimations d'erreurs avec un reste exponentiel [24] et sont d'une grande efficacité pour de fortes oscillations, mais ne permettent pas de traiter toutes les amplitudes d'oscillations. Afin d'obtenir des méthodes numériques traitant simultanément le cas de fortes et faibles oscillations, des méthodes numériques spécifiques, appelées uniformément précises (UA, de l'anglais *Uniformly accurate*) [21] ont été mises au point et reposent sur la décomposition *slow-fast*, permettant ainsi de s'affranchir du reste exponentiel en utilisant deux principales stratégies: la décomposition Micro-Macro, spécifiquement adaptée pour la précision et l'ordre élevé, ainsi que la méthode Pullback, conçue pour préserver les propriétés géométriques des équations. Ces méthodes UA réutilisent des méthodes numériques adaptées aux équations différentielles non oscillantes.

Afin de préserver les propriétés géométriques associées aux équations fortement oscillantes, une première stratégie possible est de jouer sur les propriétés des champs de vecteurs, par exemple en ajoutant de la structure hamiltonienne ou symplectique aux champs de vecteurs associés aux équations hamiltoniennes [19, 21, 24, 25]. Une autre méthode est l'utilisation directe de méthodes adaptées à la préservation de ces propriétés géométriques [23]. La dernière approche, basée sur la représentation de la solution sous forme d'une série de fonctions, permet d'obtenir une conservation d'énergie [49].

Concernant les équations aux dérivées partielles présentant un terme générant de fortes oscillations, les propriétés de la décomposition *slow-fast* peuvent se généraliser à ces EDP [19] où des schémas numériques UA peuvent être implémentés en réutilisant des schémas numériques pour les EDP d'évolution [3, 21].

2.2.2 Définitions et premières propriétés

Afin de faciliter l'étude d'une équation de la forme (20), nous devons introduire certains objets mathématiques et observer d'abord les propriétés élémentaires des solutions des équations de la forme (20).

2.2.2.1 Champ moyen

Définition 13. ***Champ moyen.*** *Soit* $f : \mathbb{R} \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$, $2\pi-$*périodique par rapport à sa première variable. Notons* $\langle f \rangle : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ *le champ moyen défini pour tout* $y \in \mathbb{R}^d$ *par :*

$$\langle f \rangle(y) = \frac{1}{2\pi} \int_0^{2\pi} f(\tau, y) \mathrm{d}\tau \tag{21}$$

Lorsque nous effectuons des simulations, nous pouvons observer que y^ε est une perturbation de la solution z du système moyen, c'est-à-dire:

$$\begin{cases} \dot{z}(t) &= \langle f \rangle (z(t)) \\ z(0) &= y^\varepsilon(0) \end{cases} \tag{22}$$

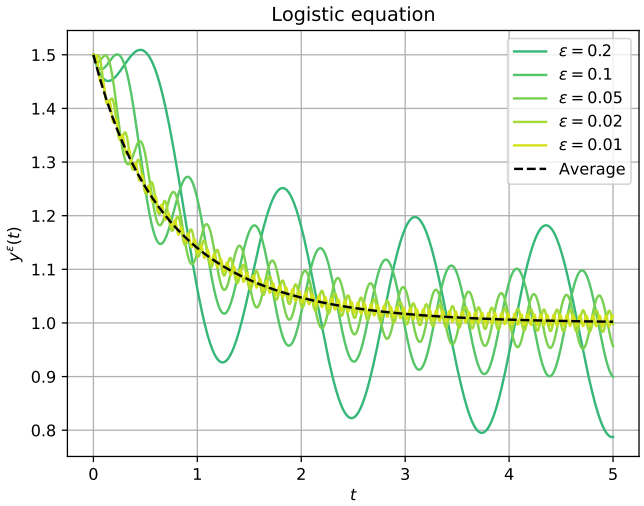

Figure 6 – Voici un exemple pour l'équation logistique : $f(\tau, y) = y(1-y) + \sin(\tau)$. Tracé des solutions pour différentes valeurs de ε. Nous observons que lorsque $\varepsilon \longrightarrow 0$, $\tilde{y}_\varepsilon$ se rapproche de plus en plus de z, solution de $\dot{z} = z(1-z)$ (ligne en pointillé).

Théorème 14. *Soit $T > 0$. Il existe une constante $C_T > 0$ telle que, pour tout $t \in [0, T]$,*

$$|y^\varepsilon(t) - z(t)| \leqslant C_T \varepsilon \tag{23}$$

Maintenant, nous connaissons une première approximation des solutions des EDO fortement oscillantes, mais si nous voulons être plus précis, nous devons étudier des propriétés supplémentaires.

2.2.2.2 Décomposition en deux dynamiques Lorsque les solutions sont tracées, on peut observer que la solution $y^\varepsilon(t)$ présente une structure organisée, composée de deux éléments principaux :

- Une tendance globale (drift), qui correspond à l'échelle longue (variable lente) dont la trajectoire est proche de $t \longmapsto z(t)$.
- Des oscillations dont la vitesse et l'amplitude et la fréquence sont contrôlées par le paramètre ε.

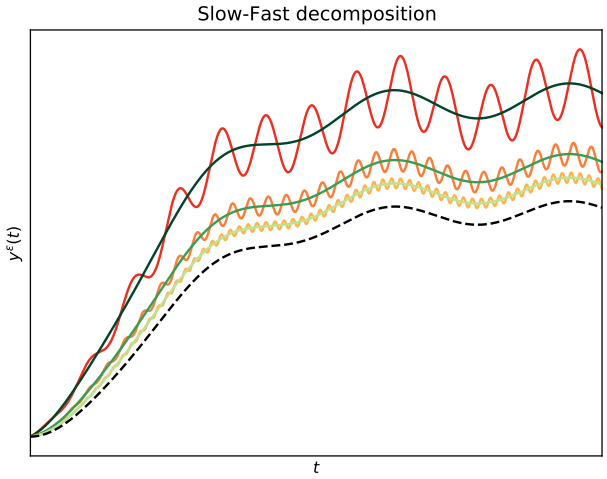

Figure 7 – Illustration de la décomposition en deux dynamiques. Le drift est représenté en vert (les teintes les plus sombres correspondent à des valeurs élevées de ε) et les oscillations supplémentaires sont représentées en teintes orangées (le rouge correspond aux valeurs élevées de ε).

2.2.2.3 Champ moyenné et changement de variable La décomposition en deux dynamiques peut être observée de cette manière:

- Le drift est la solution d'une équation autonome, non raide, appelée **équation moyennée**.
- Les fortes oscillations sont générées par une application proche de l'identité, appelée **changement de variable**.

On peut voir y^ε comme la décomposition suivante

$$y^\varepsilon(t) = \phi^\varepsilon_{\frac{t}{\varepsilon}}\left(\varphi_t^{F^\varepsilon}(y^\varepsilon(0))\right) \tag{24}$$

où $F^\varepsilon = \langle f\rangle + \mathcal{O}(\varepsilon)$ est le champ moyen et $\phi^\varepsilon = \mathrm{Id} + \mathcal{O}(\varepsilon)$ est le changement de variable, $\tau \longmapsto \phi^\varepsilon_\tau$ est 2π-périodique par rapport à τ. De plus, nous pouvons choisir $\phi^\varepsilon_0 = \mathrm{Id}$ (moyennisation stroboscopique). Pour tout $(\tau, y) \in \mathbb{R}\times\mathbb{R}^d$, $F^\varepsilon(y)$ et $\phi^\varepsilon_\tau(y)$ s'écrivent comme des séries formelles par rapport à ε. Les deux fonctions sont supposées être lisses par rapport à y et τ.

2.2.2.4 Méthode d'approximation du champ moyenné et du changement de variable Une différentiation de la formule (24) par rapport à t donne

$$\frac{\partial\phi^\varepsilon_\tau}{\partial\tau}(y) + \frac{\partial\phi^\varepsilon_\tau}{\partial y}(y)F^\varepsilon(y) = f\left(\tau, \phi^\varepsilon_\tau(y)\right). \tag{25}$$

Après intégration et moyennisation sur $[0, 2\pi]$ de la formule précédente, on obtient

$$\left\{\begin{array}{rcl} \phi^\varepsilon_\tau(y) & = & y + \displaystyle\int_0^\tau \left[f\left(\sigma, \phi^\varepsilon_\sigma(y)\right) - \frac{\partial\phi^\varepsilon_\sigma}{\partial y}(y)F^\varepsilon(y)\right]\mathrm{d}\sigma \\ F^\varepsilon(y) & = & \left(\frac{\partial\langle\phi^\varepsilon_\cdot\rangle}{\partial y}(y)\right)^{-1}\langle f\left(\cdot, \phi^\varepsilon_\cdot(y)\right)\rangle. \end{array}\right. \tag{26}$$

Cette formule donne une méthode itérative pour calculer des approximations de F^ε et ϕ^ε (qui sont des séries formelles). Considérons $F^{[0]} = \langle f\rangle$ et $\phi^{[0]} = \mathrm{Id}$. Ensuite, pour tout $n \in \mathbb{N}$, considérons

$$\left\{\begin{array}{rcl} \phi^{[n+1]}_\tau(y) & = & y + \displaystyle\int_0^\tau \left[f\left(\sigma, \phi^{[n]}_\sigma(y)\right) - \frac{\partial\phi^\varepsilon_\sigma}{\partial y}(y)F^{[n]}(y)\right]\mathrm{d}\sigma \\ F^{[n]}(y) & = & \left(\frac{\partial\langle\phi^{[n]}_\cdot\rangle}{\partial y}(y)\right)^{-1}\left\langle f\left(\cdot, \phi^{[n]}_\cdot(y)\right)\right\rangle. \end{array}\right. \tag{27}$$

Théorème 15 (Propriétés géométriques)**.** *Les propriétés suivantes sont satisfaites:*

- ***Structure Hamiltonienne :*** *Si f est Hamiltonien, c'est-à-dire s'il existe $H : (\tau, y) \longmapsto H(\tau, y)$ tel que $f(\tau, y) = \nabla_y H(\tau, y)$, alors, pour tout $n \in \mathbb{N}$, $F^{[n]}$ est Hamiltonien et $y \longmapsto \phi_\tau^{[n]}(y)$ est une application symplectique.*
- ***Divergence nulle:*** *Si f est de divergence numlle, c'est-à-dire si $\nabla_y \cdot f(\tau, y) = 0$, alors, pour tout $n \in \mathbb{N}$, $F^{[n]}$ est également de divergence nulle et $y \longmapsto \phi_\tau^{[n]}(y)$ préserve le volume (i.e. sa différentielle en espace est de déterminant 1).*

2.2.3 Décomposition en deux dynamiques avec reste exponentiel

Avec ces approximations à l'ordre désiré, nous pouvons donner une borne d'erreur pour la décomposition en deux dynamiques [24]:

Théorème 16. *Soit $T > 0$.Il existe $\varepsilon_0 > 0$ tel que, pour tout $\varepsilon \in]0, \varepsilon_0[$, il existe $n_\varepsilon \in \mathbb{N}^*$ tel que pour tout $t \in [0, T]$:*

$$\left| y^\varepsilon(t) - \phi_{\frac{t}{\varepsilon}}^{[n_\varepsilon]} \left(\varphi_t^{F^{[n_\varepsilon]}}(y^\varepsilon(0)) \right) \right| \leqslant M e^{-\frac{\beta \varepsilon_0}{\varepsilon}} \tag{28}$$

où $M, \beta > 0$ sont des constantes indépendantes de ε et de ε_0.

Cette borne d'erreur montre que la décomposition en deux dynamiques est correcte pour de petites valeurs de ε, ce qui correspond à l'étude de l'équation avec de fortes oscillations.

2.2.4 Cas autonome fortement oscillant

Un cas particulier des équations différentielles fortement oscillantes concerne le cas autonome, c'est-à-dire l'équation

$$\dot{y^\varepsilon}(t) = \frac{1}{\varepsilon} A y^\varepsilon(t) + g(y^\varepsilon(t)) \tag{29}$$

où $A \in \mathcal{M}_d(\mathbb{R})$ est une matrice dont les valeurs propres appartiennent à $i\mathbb{Z}$ et $g : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ est une fonction régulière, pas forcément linéaire.

En réalisant le changement de fonction

$$z^\varepsilon(t) = e^{-\frac{t}{\varepsilon} A} y^\varepsilon(t), \tag{30}$$

on peut alors se ramener à l'équation

$$\dot{z}^\varepsilon(t) = e^{-\frac{t}{\varepsilon}A} g\left(e^{\frac{t}{\varepsilon}A} z^\varepsilon(t)\right), \tag{31}$$

ce qui correspond bien à la forme (20).

De plus, il est possible d'obtenir une décomposition en deux dynamiques directement à partir de la forme autonome [19]:

Théorème 17. *Il existe $A^\varepsilon \in \mathcal{M}_d(\mathbb{R})$ et $g^\varepsilon : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ tels que:*

(i) *A^ε génère un flot $2\pi-$périodique $\tau \longmapsto \phi_\tau^\varepsilon = e^{\tau A^\varepsilon}$ (dynamique rapide).*

(ii) *g^ε génère un flot $t \longmapsto \varphi_t^{g^\varepsilon}$ (dynamique lente).*

(iii) *Les deux flots commutent: $\phi^\varepsilon_{\frac{t}{\varepsilon}} \circ \varphi_t^{g^\varepsilon} = \varphi_t^{g^\varepsilon} \circ \phi^\varepsilon_{\frac{t}{\varepsilon}}$.*

(iv) *Pour tous $t \in [0,T], \varepsilon \in]0,1]$, on a:*

$$\left| y^\varepsilon(t) - \phi^\varepsilon_{\frac{t}{\varepsilon}}\left(\varphi_t^{g^\varepsilon}(y_0)\right)\right| \leqslant C_T e^{-\frac{C_T}{\varepsilon}}, \tag{32}$$

où $t \longmapsto y^\varepsilon(t)$ est la solution de (29) et $C_T > 0$ est une constante indépendante de ε.

2.2.5 Méthodes UA

Pour approcher les solutions des EDO fortement oscillantes, les méthodes classiques dérivées de celles utilisées pour les EDO autonomes ne sont pas adaptées. En effet, les fortes oscillations entraînent une augmentation de l'erreur d'approximation. Par exemple, si nous voulons approximer la solution d'une équation de la forme (20) avec la méthode d'Euler explicite, nous pouvons considérer le schéma suivant pour une taille de pas $h > 0$:

$$\left\{ \begin{array}{rcl} y_{n+1} & = & y_n + hf\left(\frac{t_n}{\varepsilon}, y_n\right) \\ y_0 & = & y^\varepsilon(0), \end{array} \right. \tag{33}$$

où $t_n = nh$. Une analyse d'erreur classique montre que

$$\underset{0\leqslant n\leqslant N}{Max} |y_n - y^\varepsilon(t_n)| \leqslant \frac{Ch}{\varepsilon} \tag{34}$$

pour une certaine constante C indépendante de ε et de h. Si nous utilisons une méthode standard, les fortes oscillations entraînent une augmentation de la borne

d'erreur.

Ainsi, des méthodes numériques spécifiques, adaptées aux équations fortement oscillantes, ont été créées. Ces méthodes, appelées *méthodes uniformément précises (UA)* [21], fournissent une borne d'erreur uniforme par rapport à ε. Ces méthodes transforment l'équation (20) en équations non raides afin de supprimer l'augmentation de l'erreur due à ε.

2.2.5.1 Décomposition Micro-macro La décomposition *Micro-macro* transforme l'équation (20) en un système d'équations donné par

$$\left\{\begin{array}{rcl} \dot{v}(t) & = & F^{[n]}(v(t)) \\ \dot{w}(t) & = & f\left(\frac{t}{\varepsilon}, \phi^{[n]}_{\frac{t}{\varepsilon}}(v(t)) + w(t)\right) - \left(\frac{1}{\varepsilon}\partial_\tau \phi^{[n]}_{\frac{t}{\varepsilon}}(v(t)) + \partial_y \phi^{[n]}_{\frac{t}{\varepsilon}}(v(t)) F^{[n]}(v(t))\right) \\ (v,w)(0) & = & (y^\varepsilon(0), 0) \end{array}\right. \tag{35}$$

où $n \in \mathbb{N}$, $\phi^{[n]}$ et $F^{[n]}$ sont calculés via les formules (27). La solution y^ε est donnée par

$$y^\varepsilon(t) = \phi^{[n]}_{\frac{t}{\varepsilon}}(v(t)) + w(t). \tag{36}$$

Théorème 18. ***Estimation d'erreur - décomposition Micro-Macro.*** *Réécrivons l'équation* (35) *en*

$$\left\{\begin{array}{rcl} \dot{W}(t) & = & G^{[n]}_\varepsilon(\frac{t}{\varepsilon}, W(t)) \\ W(0) & = & (y^\varepsilon(0), 0), \end{array}\right. \tag{37}$$

où $W = (v, w)$. *Soit* Φ_h *une méthode numérique d'ordre* p. *Pour* $k \in [\![0, N]\!]$ *où* N *est le nombre d'itérations* ($h = \frac{T}{N}$ *où* T *est le temps d'intégration), notons* $(v_k, w_k) = \left(\Phi_h^{G^{[n]}_\varepsilon}\right)^k (W(0))$. *Alors, si* $n \geqslant p$, *on a*

$$\left|\phi^{[n]}_{\frac{t_k}{\varepsilon}}(v_k) + w_k - y^\varepsilon(t_k)\right| \leqslant Ch^p \tag{38}$$

où $t_k = kh$ *et* $C > 0$ *est une constante indépendante de* ε *et de* h.

Remarque. *Pour pouvoir appliquer correctement le théorème, des propriétés de régularité supplémentaires sur* f *sont nécessaires [21].*

2.2.5.2 Schéma Micro-macro integral Afin d'utiliser une approximation plus simple du changement de variable que celle requise dans le Théorème 18, nous pouvons utiliser le schéma intégral. Si nous remplaçons $G_\varepsilon^{[n]}$ par $G_\varepsilon^{[n-1]}$ dans l'équation (37) et utilisons un schéma intégral (du même ordre) en remplaçant l'évaluation par rapport à t par l'intégration, alors nous obtenons la même estimation d'erreur dans le théorème 18.

Par exemple, si nous utilisons le schéma du point milieu intégral

$$W_{k+1} = W_k + \int_{t_k}^{t_{k+1}} G_\varepsilon^{[1]}\left(\frac{t}{\varepsilon}, \frac{W_k + W_{k+1}}{2}\right) \mathrm{d}t, \tag{39}$$

alors nous obtenons

$$\underset{0\leqslant k\leqslant N}{Max} \left|\phi^{[1]}_{\frac{t_k}{\varepsilon}}(v_k) + w_k - y^\varepsilon(t_k)\right| \leqslant Ch^2. \tag{40}$$

Un cas spécifique concerne le schéma d'Euler explicite. Comme $G_\varepsilon^{[0]}$ est donné par

$$G^{[0]}(\tau, v, w) = \begin{bmatrix} \langle f\rangle(v) \\ f(\tau, v+w) - \langle f\rangle(v) \end{bmatrix}, \tag{41}$$

on peut simplifier l'écriture de ce schéma en

$$y_{k+1} = y_k + \int_{t_k}^{t_{k+1}} f\left(\frac{t}{\varepsilon}, y_k\right) \mathrm{d}t, \tag{42}$$

et ainsi nous obtenons

$$\underset{0\leqslant k\leqslant N}{Max} |y_k - y^\varepsilon(t_k)| \leqslant Ch. \tag{43}$$

2.2.5.3 Méthode Pullback la méthode *Pullback* transforme l'équation (20) en l'équation suivante

$$\begin{cases} \dot v(t) &= \left(\partial_y \phi^{[n]}_{\frac{t}{\varepsilon}}(v(t))\right)^{-1} \left(f\left(\frac{t}{\varepsilon}, \phi^{[n]}_{\frac{t}{\varepsilon}}(v(t))\right) - \frac{1}{\varepsilon}\partial_\tau \phi^{[n]}_{\frac{t}{\varepsilon}}(v(t))\right) \\ v(0) &= y^\varepsilon(0) \end{cases} \tag{44}$$

où $n \in \mathbb{N}$, $\phi^{[n]}$ est calculé via la formule (27). La solution y^ε est donnée par

$$y^{\varepsilon}(t) = \phi^{[n]}_{\frac{t}{\varepsilon}}(v(t)). \tag{45}$$

Théorème 19. ***Estimation d'erreur - Méthode Pullback.*** *Soit Φ_h une méthode numérique d'ordre p. Pour $k \in [\![0, N]\!]$ où N est le nombre d'itérations ($h = \frac{T}{N}$ où T est le temps d'intégration), notons $v_k = \left(\Phi_h^{G_\varepsilon^{[n]}}\right)^k (v(0))$. Alors, si $n \geqslant p$, nous avons :*

$$\left|\phi^{[n]}_{\frac{t_k}{\varepsilon}}(v_k) - y^{\varepsilon}(t_k)\right| \leqslant Ch^p \tag{46}$$

où $t_k = kh$ et $C > 0$ est une constante indépendante de ε et de h.

Remarques. ***(i)*** *Des propriétés supplémentaires de régularité sont nécessaires pour utiliser correctement le théorème [21].*

(ii) *La méthode Pullback peut être utilisée pour préserver les propriétés géométriques.*

2.3 Réseaux de neurones

Un domaine largement utilisé durant cette thèse concerne les réseaux de neurones [14] ainsi que leur capacité à approcher des champs de vecteurs. En particulier, nous allons donner quelques résultats et outils dans cette partie concernant les réseaux de neurones et l'entraînement, ainsi que les différentes architectures utilisées au cours de cette thèse.

2.3.1 Réseaux de neurones et équations différentielles: précédents travaux

Afin d'approcher des fonctions inconnues, une stratégie possible peut être l'utilisation de réseaux de neurones, une classe de fonctions possédant des paramètres appelés poids. Sur le même principe que la régression polynomiale, ces poids sont ajustés par minimisation d'une fonction d'erreur [28, 39].

2.3.1.1 Réseaux de neurones profonds Dans le cas des équations différentielles, un problème classique qui se pose est la recherche du champ de vecteurs associé à l'équation à partir de solutions connues en certains temps, ce qui constitue une classe de problèmes inverses. Afin d'étudier ce problème, une possibilité est

l'usage de réseaux de neurones.

La structure de base du réseau de neurones est le perceptron multicouche (MLP, de l'anglais *Multi-Layer Perceptron*), qui consiste en un empilement de fonctions affines et de fonctions non-linéaires possédant des paramètres appelés poids, le tout sur des couches [14]. Lorsqu'il y a plus d'une couche, on parle de réseau de neurones profond (DNN, de l'anglais *Deep Neural Network*).

Une première application est l'apprentissage de l'équation, c'est-à-dire du champ de vecteurs associé à l'équation, à partir d'une méthode numérique ainsi que de données provenant d'expériences [93] éventuellement bruitées [43], ce qui permet de reconstruire la dynamique associée à l'équation à partir de solutions données en certains temps. L'utilisation de l'analyse rétrograde, qui exploite les notions de champ et d'équation modifiés par rapport à une méthode numérique, permet de mener une analyse d'erreur suite à l'apprentissage [117, 118].

Lorsque l'équation différentielle possède des propriétés géométriques particulières, il est tout-à-fait possible d'encoder ces propriétés dans des réseaux de neurones afin qu'ils adoptent ces mêmes propriétés. L'exemple principal concerne les systèmes hamiltoniens, où le champ de vecteurs inconnu associé à l'équation est hamiltonien, ce qui sera alors le cas du réseau de neurones chargé de l'apprendre, donnant ainsi des réseaux de neurones hamiltoniens (HNN's, de l'anglais *Hamiltonian Neural Network*) éventuellement couplés à des méthodes symplectiques [32, 44, 82, 83, 85] ainsi que son dérivé, le réseau de neurones de Poisson (PNN, de l'anglais *Poisson Neural Network*), qui est un HNN doublé d'un auto-encodeur chargé d'apprendre un difféomorphisme [58]. Une deuxième application est l'étude de systèmes différentiels dont le champ de vecteurs associé est à divergence nulle [119] (VPNN, de l'anglais *Volume-Preserving Neural Network*), où le réseau de neurones est construit de telle sorte que sa divergence est nulle. Un autre cas de figure est celui où le champ de vecteurs associé à l'équation présente un Lagrangien, il est alors possible d'utiliser un LNN [30] (de l'anglais *Lagrangian Neural Network*), chargé d'apprendre le Lagrangien. Enfin, lorsque le flot associé à une équation différentielle reste sur une variété, il est également possible d'apprendre le champ de vecteurs associé à l'équation tout en garantissant cette appartenance du flot à la variété [110].

Structure de l'équation	-	Hamiltonienne	Poisson	Lagrangienne	Divergence nulle
Structure du réseau	DNN	HNN	PNN	LNN	VPNN

Figure 8 – Structures de réseaux de neurones en fonction de la structure du champ de vecteurs associé à une équation différentielle.

Par ailleurs, la structure en plusieurs couches des DNN's, parfois nombreuses, permet d'assimiler un DNN à la discrétisation numérique d'une équation différentielle [27, 73], ce qui permet ainsi un apprentissage de trajectoires en assimilant le réseau de neurones à un flot discrétisé [108].

De plus, une alternative au réseau de neurones est l'utilisation de la régression symbolique afin d'identifier le champ de vecteurs associé à l'équation différentielle comme appartenant à un espace vectoriel généré par une base de fonctions [11, 56], ce qui constitue ici une méthode de Machine Learning, mais en utilisant un autre espace d'approximation de fonctions que les réseaux de neurones, et où les poids jouent ici le rôle de coefficients dans la combinaison linéaire de fonctions.

2.3.1.2 PINN's En complément des perceptrons multicouche et auto-encodeurs, il existe une autre structure qui diffère des précédentes: le PINN's (de l'anglais *Physics-Informed Neural Network*), structure consistant en l'insertion directe de l'équation ou de sa structure ainsi que des conditions aux limites et initiales dans la fonction *Loss* [64, 94, 103].

Particulièrement efficace pour la résolution d'EDP's, cette structure ne nécessite pas de collecte de données si l'on cherche à résoudre une équation différentielle (c'est aussi le cas pour une EDP), et permet un apprentissage de la solution au moyen d'un réseau de neurones (le plus souvent un perceptron), de manière directe [70, 115], ou bien en passant par une méthode numérique existante [104].

De plus, cette architecture permet de tenir compte de propriétés liées à l'équation, comme par exemple une structure multi-échelle, ce qui nécessite une décomposition de la solution en deux parties [107, 109, 114], un comportement asymptotique [60] ou bien une solution à variables séparées [29]. De plus, des propriétés géométriques peuvent également être insérées dans la fonction *Loss* associée aux PINN's. En effet, il est par exemple possible de forcer la solution à conserver l'Hamiltonien [52].

De plus, il est également possible d'utiliser une autre classe de fonction que des perceptrons afin d'apprendre la solution, comme par exemple des polynômes [103]. Ainsi, dans ce cas de figure, les poids jouent le rôle de coefficients de polynômes à ajuster.

2.3.2 Définitions et propriétés d'approximation

Définition 20. ***Perceptron Multi-Couche (MLP)***

Soient $L \in \mathbb{N}^*$ et $d_0, d_1, \cdots, d_{L+1} \in \mathbb{N}^*$. Un perceptron multicouche est une application

$$\begin{array}{rccl} \mathcal{N}: & \mathbb{R}^{d_0} & \longrightarrow & \mathbb{R}^{d_{L+1}} \\ & x & \longmapsto & (A_{L+1} \circ \Sigma_L \circ A_L \circ \cdots \circ A_2 \circ \Sigma_1 \circ A_1)(x) \end{array} \tag{47}$$

*où, pour tous $1 \leqslant j \leqslant L+1$, A_j est de la forme $x \mapsto W_j x + b_j$, avec $W_j \in \mathcal{M}_{d_{j-1},d_j}(\mathbb{R})$ est la matrice des **poids** et $b_j \in \mathbb{R}^{d_j}$ est le **biais**. $\Sigma_j(x) = (\sigma(x_1), \cdots, \sigma(x_{d_j}))$ est une **fonction d'activation**. L est le **nombre de couches** du réseau de neurones. Le réseau est dit **profond** si $L > 1$.*

Remarque. *Dans la plupart des cas, σ est la fonction tangente hyperbolique, la fonction ReLu ($x \mapsto Max(0, x)$) ou bien la fonction sigmoïde ($x \longmapsto \frac{1}{1+e^{-x}}$).*

L'idée principale avec un réseau de neurones est d'ajuster ses poids ainsi que ses biais afin d'obtenir un champ de vecteurs souhaité. Des conditions d'approximation existent pour les réseaux de neurones.

Théorème 21. ***Théorème d'approximation universelle, Cybenko, 1989 [31, 88].*** *Soient $\varepsilon > 0$, $K \subset \mathbb{R}^d$ un compact, σ une fonction d'activation continue non polynomiale, et $f : \mathbb{R}^d \to \mathbb{R}^p$ une fonction continue. Alors il existe un*

réseau de neurones $\mathcal{N}$ à une couche tel que

$$||\mathcal{N} - f||_{L^\infty(K)} \leqslant \varepsilon \tag{48}$$

Théorème 22. ***Théorème d'approximation universelle pour une profondeur arbitraire [61].*** *Soient $\varepsilon > 0$, $K \subset \mathbb{R}^d$ un compact, σ une fonction d'activation continue non affine, et $f : \mathbb{R}^d \to \mathbb{R}^p$ une fonction continue. Alors il existe un réseau de neurones $\mathcal{N}$ de profondeur assez grande, tel que chaque couche possède $d + p + 2$ neurones, vérifiant*

$$||\mathcal{N} - f||_{L^\infty(K)} \leqslant \varepsilon \tag{49}$$

Remarques. ***(i)*** *Des conditions sur la profondeur ou sur chaque couche peuvent exister si l'on souhaite avoir des bornes explicites (approximation quantitative) [1].*

(ii) *D'autres bornes existent si l'on étudie l'approximation en changeant de topologie (i.e. en prenant une autre norme que la norme associée à la convergence uniforme) [4, 33].*

2.3.3 Approximation d'une fonction avec un réseau de neurones

Considérons une fonction inconnue $f : \mathbb{R}^d \longrightarrow \mathbb{R}^p$. Notre objectif est d'approcher f avec un réseau de neurones. Notons $f_\theta : \mathbb{R}^d \longrightarrow \mathbb{R}^p$ le réseau de neurones qui devra approcher f, où θ correspond aux paramètres du réseau de neurones (poids et biais).

Le processus d'approximation de f par f_θ peut être réalisé en suivant deux étapes principales: la collecte de données et l'entraînement.

2.3.3.1 Collecte de données et entraînement

1 **Collecte de données:** Tout d'abord, pour avoir de l'information concernant f, nous avons besoin de données sur cette fonction inconnue. Pour cela, nous collectons un ensemble de données composé de K données décrites par $\{(x_k, y_k)\}_{0 \leqslant k \leqslant K-1}$, où, pour tout $k \in [\![0, K-1]\!]$, le couple de vecteurs $(x_k, y_k) \in \mathbb{R}^d \times \mathbb{R}^p$ satisfait $y_k = f(x_k)$.

2 **Approximation par réseau de neurones:** Ensuite, si f_θ doit approcher f, nous aimerions avoir, pour tout $k \in [\![0, K-1]\!]$, $y_k \approx f_\theta(x_k)$. Par conséquent,

en considérant $K_0 \in [\![1, K-1]\!]$, nous ajustons les paramètres θ du réseau de neurones afin d'avoir la meilleure appproximation possible sur les K_0 premières données. Les autres données sont utilisées afin de vérifier que l'entraînement (également appelé apprentissage) fonctionne bien, et que le réseau de neurones ainsi entraîné est capable de se généraliser à d'autres données [14].

Pour réaliser cette étape, nous introduisons les fonctions *Loss*.

Définition 23. ***Fonctions* Loss - *(MSE)*** *On définit la fonction d'Erreur Moyenne Quadratique (MSE, de l'anglais* Mean Squared Error*) de perte, appelée* Loss *sur les données d'entraînement et de test par:*

$$Loss_{Train} = \frac{1}{K_0} \sum_{k=0}^{K_0-1} |y_k - f_\theta(x_k)|^2 \tag{50}$$

$$Loss_{Test} = \frac{1}{K - K_0} \sum_{k=K_0}^{K-1} |y_k - f_\theta(x_k)|^2 \tag{51}$$

Afin d'entraîner correctement f_θ, on utilise un algorithme d'optimisation afin de minimiser la fonction $Loss_{Train}$ par rapport aux paramètres θ. Pendant que l'on minimise la fonction $Loss_{Train}$, on observe l'évolution de la fonction $Loss_{Test}$ avec les mêmes paramètres calculés θ. Par exemple, si l'on utilise une descente de gradient à pas fixe, les itérations successives sont données par

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\partial Loss_{Train}}{\partial \theta}(\theta_t) \tag{52}$$

Si les suites $(Loss_{Train}(\theta_t))_{t \geqslant 0}$ et $(Loss_{Test}(\theta_t))_{t \geqslant 0}$ ont la même décroissance, alors l'entraînement est correctement fait. Au contraire, un sur-apprentissage est détecté par une suite $(Loss_{Test}(\theta_t))_{t \geqslant 0}$ qui ne décroît pas, signe que le réseau de neurones entraîné ne peut pas généraliser à d'autres données. De plus, un sous-apprentissage peut être détecté par une mauvaise décroissance des deux suites.

Problème	Cause	Effet sur les données d'entraînement	Effet sur les données de test
Sous-apprentissage	Modèle trop simple par rapport au nombre de données	Mauvaise performance	Mauvaise performance
Sur-apprentissage	Modèle trop complexe par rapport au nombre de données	Très bonne performance	Mauvaise performance

Figure 9 – Causes et effets d'un sous/sur-apprentissage lors de l'entraînement. Le modèle désigne ici le réseau de neurones. il est dit simple s'il a peu de paramètres et complexe si au contraire il en possède beaucoup (nombre de couches et de neurones).

2.3.3.2 Algorithme de descente de gradient

Afin de minimiser les fonctions *Loss*, l'algorithme de descente de gradient à pas fixe n'est pas suffisant. L'une des raisons de cette inefficacité est la non-convexité des fonctions *Loss*. Afin d'optimiser des fonctions non-convexes, des algorithmes plus efficaces sont nécessaires.

Afin d'avoir un apprentissage efficace, on utilise l'algorithme *Adam* [28], qui utilise les premier et second moments du gradient de la fonction *Loss* afin d'adapter les itérations de la descente et d'avoir une optimisation de meilleure qualité.

- **Entrées:** α (Pas de descente - *learning rate*), β_1, β_2, θ_0 (paramètres liés aux moments), $\theta \mapsto Loss_{Train}(\theta)$ (*Loss* function), λ (poids de descente - *weight decay*), ε (paramètre de régularisation).
- **Initialisation:** $m_0 = 0$ (premier moment), $v_0 = 0$ (second moment).
- **Algorithme:** pour $t \in [\![0, N_{epochs} - 1]\!]$

$$\begin{aligned} g_{t+1} &= \frac{\partial Loss_{Train}}{\partial \theta}(\theta_t) + \lambda \theta_t \\ m_{t+1} &= \beta_1 m_t + (1-\beta_1) g_{t+1} \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2) g_{t+1}^2 \\ \widehat{m_{t+1}} &= \frac{m_{t+1}}{1-\beta_1^t} \\ \widehat{v_{t+1}} &= \frac{v_{t+1}}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \alpha \frac{\widehat{m_{t+1}}}{\sqrt{\widehat{v_{t+1}}} + \varepsilon} \end{aligned} \tag{53}$$

De plus, afin d'obtenir une optimisation plus rapide (en particulier pour le calcul de la *Loss*) et plus précise, on utilise les petits lots (*mini-batching*). Une méthode classique, très courante dans le cas de la descente de gradient à pas fixe et donnant l'algorithme du gradient stochastique (SGD, de l'anglais *Stochastic Gradient Descent)*, consiste à n'effectuer l'itération de l'algorithme que sur un petit lot, autrement dit, à chaque itération, on choisit aléatoirement un échantillon $\mathcal{B}$ de données parmi celles utilisées pour l'entraînement. Ensuite, dans l'algorithme de descente de gradient, on remplace $\frac{\partial Loss_{Train}}{\partial \theta}$ par $\frac{\partial Loss_{Train,\mathcal{B}}}{\partial \theta}$ où $Loss_{Train,\mathcal{B}}$ est donné par

$$Loss_{Train,\mathcal{B}} := \frac{1}{B}\sum_{i=1}^{B}\left|y_{b_j} - f_\theta(x_{b_j})\right|^2 \tag{54}$$

où $\mathcal{B} := \{b_1, \cdots, b_B\}$ est l'ensemble des indices des données choisies durant l'itération.

Une variante de cette méthode est d'effectuer la descente sur tous les petits lots, c'est-à-dire d'appliquer l'algorithme à la fonction *Loss* sur tous les lots de données, autrement dit en le répétant sur chaque lot à chaque itération.

2.3.4 Ajout de structure sur les réseaux de neurones

Dans de nombreux cas, on ajoute de la structure sur un réseau de neurones afin de préserver certaines propriétés ou d'effectuer des calculs d'inverse.

2.3.4.1 Perturbation d'un champ connu Dans certains cas, on veut apprendre la perturbation d'un champ. Plus précisément, si notre fonction inconnue f s'écrit selon la décomposition $f = g + R$, où g est connu, il n'est pas obligatoire d'apprendre g avec un réseau de neurones, mais seulement R en l'approchant par un réseau de neurones R_θ, et la fonction *Loss* devient alors

$$Loss_{Train} = \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|y_k - g(x_k) - R_\theta(x_k)\right|^2 \tag{55}$$

Dans de nombreux cas, un petit paramètre est situé devant le terme R, devenant ainsi une *petite* perturbation. Par exemple, si $f = g + \eta R$, où η est un petit paramètre, on peut entraîner la fonction *Loss*

$$Loss_{Train} = \frac{1}{K_0} \sum_{k=0}^{K_0-1} \left| \frac{y_k - g(x_k) - \eta R_\theta(x_k)}{\eta} \right|^2 \tag{56}$$

afin d'isoler la perturbation et d'obtenir la fonction *Loss* de $R(x_k) - R_\theta(x_k)$.

2.3.4.2 Structure d'auto-encodeur Bien qu'apprendre $f : \mathbb{R}^d \longrightarrow \mathbb{R}^p$ soit important, nous avons besoin dans certaines situations de connaître la manière de passer des données x_k à $y_k = f(x_k)$. Par conséquent, nous allons approcher f par $f_{\theta,+}$ et calculer un autre réseau de neurones $f_{\theta,-}$ tel que $x_k \approx f_{\theta,-}(y_k)$. Dans le cas $d = p$ et si f est inversible, $f_{\theta,-}$ doit approcher f^{-1}.

Afin de correctement approcher cette paire de réseaux de neurones, on optimise cette fonction *Loss*

$$\begin{aligned} Loss_{Train} &= \frac{1}{K_0} \sum_{k=0}^{K_0} |y_k - f_{\theta,+}(x_k)|^2 + \frac{1}{K_0} \sum_{k=0}^{K_0} |x_k - f_{\theta,-}(y_k)|^2 \\ &+ \frac{1}{K_0} \sum_{k=0}^{K_0} |x_k - f_{\theta,-}\left(f_{\theta,+}(x_k)\right)|^2 + \frac{1}{K_0} \sum_{k=0}^{K_0} |y_k - f_{\theta,+}\left(f_{\theta,-}(y_k)\right)|^2 \end{aligned} \tag{57}$$

De plus, dans certains cas, l'un des deux premiers termes est négligé, en particulier lorsque les auto-encodeurs sont utilisés en complément d'autres réseaux de neurones. Cependant, les deux derniers termes sont importants puisqu'ils forcent la "réciprocité" entre $f_{\theta,+}$ et $f_{\theta,-}$.

2.3.4.3 Résolution d'équations différentielles par les PINN's Afin de résoudre des équations aux dérivées partielles sur un domaine ouvert $\Omega \subset \mathbb{R}^d$ relativement compact (i.e. tel que $\overline{\Omega}$ est compact) et sur un intervalle de temps $[0, T]$, présentant la forme suivante:

$$\begin{cases} \frac{\partial u}{\partial t}(t,x) &= f\left(t, x, u(t,x), \nabla_x u(t,x), \cdots, \nabla_x^N u(t,x)\right), & (t,x) \in [0,T] \times \Omega \\ u(0,x) &= u_0(x), & x \in \Omega \\ Bu(t,\sigma) &= g(\sigma), & (t,\sigma) \in [0,T] \times \partial\Omega, \end{cases} \tag{58}$$

où f, u_0 et g sont des fonctions régulières, B est un opérateur différentiel défini sur $\partial\Omega$ et $u : [0,T] \times \Omega \longrightarrow \mathbb{R}^q$ est la solution inconnue.

On utilise pour cela des PINN's (de l'anglais *Physics-Informed Neural Networks*).

Afin d'approcher u, on l'apprend par un réseau de neurones "classique" (dans la plupart des cas un MLP) noté u_θ ($u_\theta : \mathbb{R}^{d+1} \longrightarrow \mathbb{R}$ est un perceptron multicouche, dont l'entrée est le vecteur $(t, x_{(1)}, \cdots, x_{(d)}) \in \mathbb{R}^{d+1}$) ($x_{(i)}$ est la i-ème corrdonée de x):

$$\forall (t,x) \in [0,T] \times \Omega, \quad u_\theta(t,x) \approx u(t,x). \tag{59}$$

On choisit aléatoirement K données $(t_k, x_k, \sigma_k) \in [0,T] \times \Omega \times \partial\Omega$ pour tous $0 \leqslant k \leqslant K-1$ et on garde K_0 données pour l'entraînement. Ces données correspondent à une discrétisation du domaine de résolution de l'EDP.

Ensuite, on réalise l'entraînement en minimisant la fonction $Loss_{Train}$:

$$\begin{aligned} Loss_{Train} \quad := \quad & \underbrace{\frac{1}{K_0} \sum_{k=0}^{K_0-1} \left| \frac{\partial u_\theta}{\partial t}(t_k,x_k) - f\left(t_k, x_k, u_\theta(t_k,x_k), \nabla_x u_\theta(t_k,x_k), \cdots, \nabla_x^N u_\theta(t_k,x_k)\right)\right|^2}_{\text{Equation}} \\ + \quad & \underbrace{\frac{1}{K_0} \sum_{k=0}^{K_0-1} |u_\theta(0,x_k) - u_0(x_k)|^2}_{\text{Condition initiale}} + \underbrace{\frac{1}{K_0} \sum_{k=0}^{K_0-1} |Bu_\theta(t_k,\sigma_k) - g(\sigma_k)|^2}_{\text{Condition au bord}}. \end{aligned} \tag{60}$$

On observe en même temps l'évolution de la fonction $Loss_{Test}$ sur les données K_0 à $K-1$.

Concernant les équations différentielles ordinaires, si l'on veut apprendre le flot d'une EDO de la forme suivante:

$$\begin{cases} \frac{\partial \varphi_t}{\partial t}(x) &= f\left(t, \varphi_t(x)\right), & (t,x) \in [0,T] \times \Omega \\ \varphi_0(x) &= x, & x \in \Omega, \end{cases} \tag{61}$$

où $\Omega \subset \mathbb{R}^d$ est un compact, on approche $\varphi_t(x)$ par un réseau de neurones, noté $\varphi_\theta(t,x)$ ($\varphi_\theta : \mathbb{R}^{d+1} \longrightarrow \mathbb{R}^d$ est un perceptron multicouche dont l'entrée est le vecteur

$(t, x_{(1)}, \cdots, x_{(d)})) \in \mathbb{R}^{d+1}$):

$$\forall (t,x) \in [0,T] \times \Omega, \quad \varphi_\theta(t,x) \approx \varphi_t(x). \tag{62}$$

On choisit aléatoirement K données $(t_k, x_k) \in [0,T] \times \Omega$ pour tous $0 \leqslant k \leqslant K-1$ et on garde K_0 données pour l'entraînement. Ces données correspondent à la discrétisation du domaine d'apprentissage du flot.

Ensuite, on entraîne la fonction $Loss_{Train}$, de la même forme que celle utilisée dans le cas des EDP's sans terme de condition au bord:

$$\begin{aligned} Loss_{Train} \quad := \quad & \underbrace{\frac{1}{K_0} \sum_{k=0}^{K_0-1} \left| \frac{\partial \varphi_\theta}{\partial t}(t_k, x_k) - f\left(t_k, \varphi_\theta(t_k, x_k)\right) \right|^2}_{\text{Equation}} \\ + \quad & \underbrace{\frac{1}{K_0} \sum_{k=0}^{K_0-1} \left| \varphi_\theta(0, x_k) - x_k \right|^2}_{\text{Condition initiale}}. \end{aligned} \tag{63}$$

On observe en même temps l'évolution de la fonction $Loss_{Test}$ sur les données K_0 à $K-1$.

Remarque. *On ajoute souvent un coefficient multiplicatif pour chaque terme de la fonction Loss, afin de donner plus ou moins d'importance au terme correspondant.*

3 Plan du manuscrit

Le manuscrit se divise en quatre chapitres. Un premier chapitre, consacré au cas d'équations autonomes, sert de base aux deuxième et troisième chapitres, consacrés au cas fortement oscillant. Bien qu'utilisant certains éléments du troisième chapitre, le quatrième et dernier chapitre est plus indépendant.

3.1 Chapitre 1: Machine Learning et Equations autonomes

Ce premier chapitre, adapté de l'article [10] est consacré à l'étude du cas des équations différentielles autonomes de la forme

$$\dot{y} = f(y) \tag{64}$$

où $f : \mathbb{R}^d \longmapsto \mathbb{R}^d$ est une fonction régulière. L'idée est d'améliorer des méthodes numériques existantes en utilisant du Machine Learning.

Si l'on considère une méthode numérique à un pas, notée Φ_h^f, où h est le pas de temps et Φ_h^f est le flot numérique par rapport à cette méthode en utilisant le champ de vecteurs f, supposée d'ordre p, alors, en prenant un temps d'intégration T et N itérations de la méthode numérique telles que $h = \frac{T}{N}$, il existe une constante C, indépendante de h, telle que, pour tout $h > 0$ suffisamment petit,

$$\underset{0 \leqslant n \leqslant N}{Max} \left| \varphi_{nh}^f(y(0)) - \left(\Phi_h^f\right)^n (y(0)) \right| \leqslant C h^p. \tag{65}$$

Si l'on décide de perturber le champ de vecteurs f en un champ de vecteurs appelé champ modifié tronqué à l'ordre q noté $\tilde{f}_h^{[q]}$ tel que

$$\tilde{f}_h^{[q]} = f + h^p f_1 + h^{p+1} f_2 + \cdots + h^{p+q-2} f_{q-1} \tag{66}$$

où les champs f_i ne dépendent que de la variable en espace et sont calculés à partir des dérivées successives de f. De cette manière, on a une montée en ordre possible en appliquant la méthode numérique Φ_h avec le champ $\tilde{f}_h^{[q]}$:

$$\underset{0 \leqslant n \leqslant N}{Max} \left| \varphi_{nh}^f(y(0)) - \left(\Phi_h^{\tilde{f}_h^{[q]}}\right)^n (y(0)) \right| \leqslant C' h^{p+q-1}. \tag{67}$$

Bien que calculables, les termes successifs du champ modifié tronqué s'obtiennent via des formules complexes. L'idée du premier chapitre est donc d'apprendre ces termes successifs via des réseaux de neurones. Pour réaliser cette opération, on introduit un réseau de neurones, noté $f_\theta(\cdot, h)$, chargé d'approcher $\tilde{f}_h^{[q]}$, en l'écrivant sous cette forme:

$$f_\theta(\cdot, h) = f + h^p f_{\theta,1} + \cdots + h^{p+q-2} f_{\theta,q-1} + h^{p+q-1} R_\theta(\cdot, h) \tag{68}$$

où q est le nombre de termes que l'on souhaite introduire dans le réseau de neurones, de façon à imiter la structure du champ modifié tronqué, complété d'un reste chargé d'apprendre les termes restants. Les $f_{\theta,i}$ et R_θ sont des perceptrons.

Ainsi, pour créer des données, on simule K données en calculant le flot exact (en réalité une méthode très précise mais aussi coûteuse):

$$\forall k \in [\![0, K-1]\!], \quad y_1^k = \varphi_{h^k}^f(y_0^k) \tag{69}$$

où, pour tout $k \in [\![0, K-1]\!]$, h^k est un pas de temps pris aléatoirement dans un domaine $[h_-, h_+]$ et y_0^k est une donnée initiale prise aléatoirement dans un domaine $\Omega \subset \mathbb{R}^d$, supposé compact.

Une fois ces données créées, on peut apprendre $f_\theta(\cdot, h)$ en minimisant la fonction de perte *Loss* suivante, fonction d'erreur moyenne quadratique sur les K_0 premières données:

$$Loss_{Train} = \frac{1}{K_0} \sum_{k=0}^{K_0-1} \frac{1}{h^{k^{2p+2}}} \left| y_1^k - \Phi_{h^k}^{f_\theta(\cdot,h^k)}(y_0^k) \right|^2. \tag{70}$$

Pour détecter un éventuel sur-apprentissage, on regarde également l'évolution de la $Loss_{Test}$, l'erreur quadratique moyenne sur les données restantes:

$$Loss_{Test} = \frac{1}{K - K_0} \sum_{k=K_0}^{K-1} \frac{1}{h^{k^{2p+2}}} \left| y_1^k - \Phi_{h^k}^{f_\theta(\cdot,h^k)}(y_0^k) \right|^2. \tag{71}$$

Une fois l'entraînement terminé, l'intégration numérique avec $f_\theta(\cdot, h)$ donne une erreur du même ordre que celle fournie par la méthode classique, mais avec une constante multiplicative réduite. En effet, si l'on note δ l'erreur d'apprentissage du champ modifié tronqué (ou plutôt de sa perturbation, car le premier terme, donné par f, est connu) donnée par

$$\delta := \underset{(y,h)\in[h_-,h_+]\times\Omega}{Max} \left| \frac{f_\theta(y,h) - \tilde{f}_h^{[q]}(y)}{h^p} \right|, \tag{72}$$

alors, on a l'erreur suivante

$$\underset{0\leqslant n\leqslant N}{Max} \left| \varphi_{nh}^f(y(0)) - \left(\Phi_h^{f_\theta(\cdot,h)}\right)^n (y(0)) \right| \leqslant C_\theta \delta h, \tag{73}$$

où C_θ est une constante indépendante de h.

Pour certaines valeurs de h, il est ainsi possible d'obtenir une erreur numérique

inférieure à certaines méthodes d'ordre plus élevé (ordre 5 par exemple). Des tests numériques ont été menés sur deux systèmes différentiels (Pendule simple non linéaire et système "Rigid Body") en utilisant les méthodes d'Euler explicite, de Runge-Kutta 2 et du point milieu (implicite). Pour la méthode du point milieu appliquée au pendule simple, une étude de la conservation de l'Hamiltonien a été faite.

Dans le cas de la méthode d'Euler explicite, une stratégie supplémentaire a été mise en oeuvre afin d'apprendre séparément les premiers termes du champ modifié tronqué. S'appuyant sur plusieurs simulations avec un pas de temps différent pour une même condition initiale ainsi que l'inverse de Moore-Penrose d'une matrice, cette méthode peut permettre un temps d'apprentissage plus court si l'on choisit d'apprendre simultanément les termes du champ modifié tronqué.

3.2 Chapitre 2: Amélioration d'un schéma intégral d'ordre 1 via des réseaux de Neurones

À partir de ce chapitre, nous considèrerons le cas des équations fortement oscillantes:

$$\dot{y^\varepsilon}(t) = f\left(\frac{t}{\varepsilon}, y^\varepsilon(t)\right) \tag{74}$$

où $f : \mathbb{R} \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ est une fonction régulière, 2π-périodique par rapport à sa première variable. $\varepsilon \in]0, 1]$ est un paramètre, qui, en tendant vers 0, est responsable de fortes oscillations.

Pour résoudre numériquement ces équations différentielles, les méthodes classiques utilisées dans le cas autonome ne suffisent plus. En effet, si on considère le schéma d'Euler explicite, donné par la formule suivante:

$$\begin{cases} y_0 &= y^\varepsilon(0) \\ \forall n \in \mathbb{N}, \quad y_{n+1} &= y_n + hf\left(\frac{nh}{\varepsilon}, y_n\right) \end{cases}, \tag{75}$$

alors on a, pour N itérations, une erreur de la forme

$$\underset{0 \leqslant n \leqslant N}{Max} |y_n - y^\varepsilon(nh)| \leqslant \frac{Ch}{\varepsilon}, \tag{76}$$

où C est une constante indépendante de h et ε. L'erreur numérique se dégrade fortement lorsque ε diminue. Pour cette raison, les méthodes UA sont employées. Pour le cas de l'ordre 1, une méthode ne nécessite pas de calcul supplémentaire (à l'exception du champ moyen $\langle f \rangle$): il s'agit du schéma d'Euler intégral, donné par la formule récursive suivante:

$$\left\{ \begin{array}{rcl} y_0 & = & y^\varepsilon(0) \\ \forall n \in \mathbb{N}, \quad y_{n+1} & = & y_n + \displaystyle\int_{nh}^{(n+1)h} f\left(\frac{t}{\varepsilon}, y_n\right) \mathrm{d}t \end{array} \right. , \tag{77}$$

Cette méthode, uniformément précise en ε, permet d'obtenir une convergence du premier ordre:

$$\underset{0 \leqslant n \leqslant N}{Max} |y^\varepsilon(nh) - y_n| \leqslant C' h \tag{78}$$

où $C' > 0$ est une constante indépendante de ε et h.

Comme dans le cas autonome, on peut en fixant $\varepsilon \in]0, 1]$, trouver un champ de vecteurs afin d'effectuer une montée en ordre. En effet, si on considère $h > 0$, il existe un champ de vecteurs $R_p^\varepsilon : \mathbb{R} \times \mathbb{R} \times \mathbb{R}^d \times \mathbb{R} \longrightarrow \mathbb{R}^d$ ainsi que la suite donnée par

$$\left\{ \begin{array}{rcl} z_0 & = & y^\varepsilon(0) \\ \forall n \in \mathbb{N}, \quad z_{n+1} & = & z_n + \displaystyle\int_{nh}^{(n+1)h} f\left(\frac{t}{\varepsilon}, z_n\right) \mathrm{d}t \\ & + & h^2 R_p^\varepsilon\left(\cos\left(\frac{nh}{\varepsilon}\right), \sin\left(\frac{nh}{\varepsilon}\right), z_n, h\right) \end{array} \right. \tag{79}$$

soit une approximation de la vraie solution à l'ordre p:

$$\underset{0 \leqslant n \leqslant N}{Max} |z_n - y^\varepsilon(nh)| \leqslant C''_\varepsilon h^p. \tag{80}$$

Tout comme le champ modifié dans le cas autonome, le champ R_p^ε est lui aussi très complexe à calculer. Ainsi, le but de ce chapitre va être de l'approcher à l'aide d'un réseau de neurones

$$R_\theta^\varepsilon(u, v, y, h) \approx R_p^\varepsilon(u, v, y, h). \tag{81}$$

R_θ^ε est un perceptron multicouche. On peut donc employer du Machine Learning pour améliorer la méthode existante pour un $\varepsilon \in]0, 1]$ fixé.

Pour créer des données, on simule K données en calculant le flot exact (à l'aide d'une méthode précise mais coûteuse et non adaptée aux problèmes raides comme celui-ci), pour tout $k \in [\![0, K-1]\!]$,

$$y_1^k = \varphi_{h^k}^f \left(y_0^k \right) \tag{82}$$

où, pour tout $k \in [\![0, K-1]\!]$, h^k est un pas de temps pris aléatoirement dans un domaine $[h_-, h_+]$ et $y_0^k = y^\varepsilon(t_0^k)$ est une donnée initiale prise aléatoirement dans un domaine $\Omega \subset \mathbb{R}^d$, supposé compact. De plus, t_0^k est un temps initial pris aléatoirement dans $[0, 2\pi]$.

Une fois ces données créées, l'étape d'entraînement permet d'apprendre R_θ^ε en minimisant la fonction *Loss* suivante, fonction d'erreur moyenne quadratique sur les K_0 premières données:

$$Loss_{Train} = \frac{1}{K_0} \sum_{k=0}^{K_0-1} \frac{1}{h^{k^4}} \left| y_1^k - \hat{y_1}^k \right|^2 \tag{83}$$

où, pour tout $k \in [\![0, K]\!]$, $\hat{y_1}^k$ est donné par

$$\begin{aligned} \hat{y_1}^k &= y_0^k + \int_{t_0^k}^{t_0^k + h^k} f\left(\frac{t}{\varepsilon}, y_0^k \right) \mathrm{d}t \\ &+ h^2 R_\theta^\varepsilon \left(\cos\left(\tfrac{t_0^k}{\varepsilon} \right), \sin\left(\tfrac{t_0^k}{\varepsilon} \right), y_0^k, h^k \right). \end{aligned} \tag{84}$$

Pour détecter un éventuel sur-apprentissage, on regarde également l'évolution de la $Loss_{Test}$, l'erreur quadratique moyenne sur les données restantes:

$$Loss_{Test} = \frac{1}{K - K_0} \sum_{k=K_0}^{K-1} \frac{1}{h^{k^4}} \left| y_1^k - \hat{y_1}^k \right|^2. \tag{85}$$

Une fois l'entraînement terminé, l'intégration numérique avec R_θ^ε donne une erreur similaire à celle fournie par la méthode classique, mais, comme dans le cas autonome, à une constante multiplicative près réduite. De plus, cette constante multiplicative, du fait que ε est fixé pour chaque apprentissage, dépend de ce paramètre. Si l'on note l'erreur d'apprentissage

$$\delta_\varepsilon := \underset{(u,v,y,h) \in [-1,1] \times [-1,1] \times \Omega \times [0, h_+]}{Max} \left| R_p^\varepsilon(u, v, y, h) - R_\theta^\varepsilon(u, v, y, h) \right|, \tag{86}$$

alors il existe une constante $C_\varepsilon > 0$ telle que, si la suite $(y^\varepsilon_{\theta,n})_{n\in\mathbb{N}}$ est donnée par:

$$y^\varepsilon_{\theta,n+1} = y^\varepsilon_{\theta,n} + \int_{nh}^{(n+1)h} f\left(\frac{t}{\varepsilon}, y^\varepsilon_{\theta,n}\right) \mathrm{d}t + h^2 R^\varepsilon_\theta \left(\cos\left(\frac{nh}{\varepsilon}\right), \sin\left(\frac{nh}{\varepsilon}\right), y^\varepsilon_{\theta,n}, h\right), \quad (87)$$

alors on a, si $h = \frac{T}{N}$, où N est le nombre d'itérations:

$$\underset{0\leqslant n\leqslant N}{Max} \left|y^\varepsilon_{\theta,n} - y^\varepsilon(nh)\right| \leqslant C_\varepsilon \delta_\varepsilon h. \quad (88)$$

Pour certaines valeurs de h, il est possible d'obtenir de meilleurs résultats par rapport à un calcul direct de champ de vecteurs à l'ordre 2 R^ε_2. Des tests numériques ont été menés sur le système différentiel correspondant au Pendule inversé.

3.3 Chapitre 3: Deep Learning pour les Equations fortement oscillantes

Dans ce troisième chapitre, on considère toujours les équations fortement oscillantes de la forme

$$\dot{y^\varepsilon}(t) = f\left(\frac{t}{\varepsilon}, y^\varepsilon(t)\right), \quad (89)$$

où $f : \mathbb{R} \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ est une fonction régulière, $2\pi-$périodique par rapport à sa première variable, et $\varepsilon \in]0, 1]$.

L'objectif est ici d'exploiter la structure multi-échelle des solutions afin d'apporter une résolution numérique de ce type d'équation en s'aidant du Machine Learning, et en se basant sur les méthodes UA, en particulier la décomposition Micro-Macro.

3.3.1 Décomposition en deux dynamiques

Dans un premier temps, intéressons-nous à la décomposition en deux dynamiques de la solution $t \longmapsto y^\varepsilon(t)$, donnée (à un reste exponentiel près), pour tous $t \in [0, T]$, $\varepsilon \in]0, \varepsilon_0[$, par

$$\forall t \in [0, T], \left|y^\varepsilon(t) - \phi^{\varepsilon,[n_\varepsilon]}_{\frac{t}{\varepsilon}} \left(\varphi_t^{F^{\varepsilon,[n_\varepsilon]}}(y^\varepsilon(0))\right)\right| \leqslant M e^{-\frac{\beta\varepsilon_0}{\varepsilon}} \quad (90)$$

où $M, \beta > 0$ sont des constantes indépendantes de ε et $\varepsilon_0 \in]0, 1]$ ne dépend que de l'équation. Les champs de vecteurs $y \longmapsto F^{\varepsilon,[k]}(y)$ et $(\tau, y) \longmapsto \phi_\tau^{\varepsilon,[k]}(y)$ correspondent respectivement au champ de vecteurs associé à l'équation moyennée (qui est ainsi une équation autonome) approché à l'ordre ε^k, qui est responsable de la dynamique lente et au générateur de fortes oscillations également approché à l'ordre ε^k, qui est responsable de la dynamique rapide. Comme ces champs de vecteurs sont complexes à calculer, la stratégie est ici de les approcher à l'aide de réseaux de neurones.

Si l'on néglige le reste exponentiel, alors on a, de manière approximative, pour $t_0 \in \mathbb{R}$ et $h > 0$:

$$\begin{aligned}
y^\varepsilon(t_0+h) &\approx \phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_0+h}{\varepsilon}} \left(\varphi^{F^{\varepsilon,[n_\varepsilon]}}_{t_0+h}(y^\varepsilon(0))\right) \\
&\approx \phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_0+h}{\varepsilon}} \left(\varphi^{F^{\varepsilon,[n_\varepsilon]}}_{h} \left(\varphi^{F^{\varepsilon,[n_\varepsilon]}}_{t_0}(y^\varepsilon(0))\right)\right) \\
&\approx \phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_0+h}{\varepsilon}} \left(\varphi^{F^{\varepsilon,[n_\varepsilon]}}_{h} \circ \left(\phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_0}{\varepsilon}}\right)^{-1} \circ \left(\phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_0}{\varepsilon}}\right) \left(\varphi^{F^{\varepsilon,[n_\varepsilon]}}_{t_0}(y^\varepsilon(0))\right)\right) \\
&\approx \left(\phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_0+h}{\varepsilon}} \circ \varphi^{F^{\varepsilon,[n_\varepsilon]}}_{h} \circ \left(\phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_0}{\varepsilon}}\right)^{-1}\right)(y^\varepsilon(t_0)) \qquad (91) \\
&\approx \left(\phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_0+h}{\varepsilon}} \circ \Phi^{\widetilde{F^{\varepsilon,[n_\varepsilon]}}_h}_h \circ \left(\phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_0}{\varepsilon}}\right)^{-1}\right)(y^\varepsilon(t_0))
\end{aligned}$$

ce qui nécessite non seulement d'apprendre le champ de vecteurs $(\tau, y) \longmapsto \phi_\tau^\varepsilon(y)$, mais également son inverse (par rapport à la variable d'espace y), d'où l'utisation d'un auto-encodeur. De plus, les travaux du premier chapitre sur les équations autonomes permettent un apprentissage du champ moyenné modifié $\widetilde{F_h^{\varepsilon,[n_\varepsilon]}}$ par rapport à une méthode numérique à un pas Φ_h.

On introduit alors les réseaux de neurones F_θ, ϕ_θ et $\phi_{\theta,-}$, respectivement chargés d'apprendre $\widetilde{F_h^{\varepsilon,[n_\varepsilon]}}$, $\phi_\cdot^{\varepsilon,[n_\varepsilon]}$ et $\left(\phi_\cdot^{\varepsilon,[n_\varepsilon]}\right)^{-1}$, sous les formes suivantes:

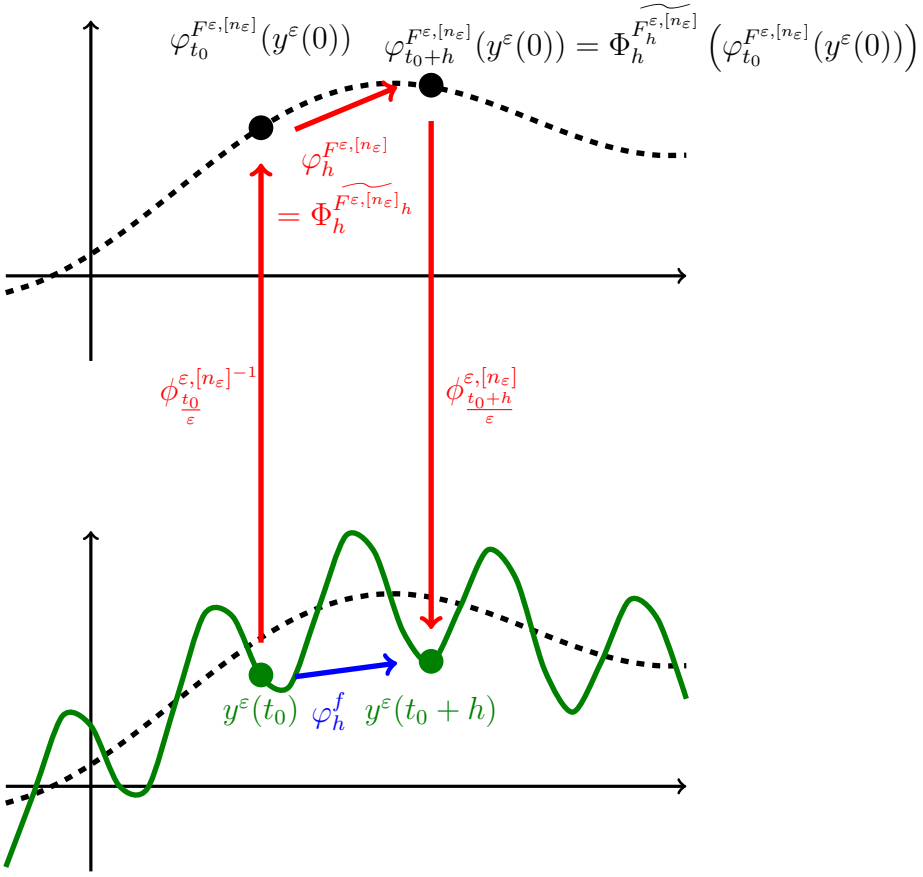

Figure 10 – Illustration de la stratégie employée durant ce chapitre afin d'apprendre le changement de variable, son inverse ainsi que le champ moyenné modifié via la formule d'approximation (91).

$$\begin{aligned} F_\theta(y,h,\varepsilon) &= \langle f\rangle + R_{\theta,F}(y,h,\varepsilon) \\ \phi_\theta(\tau,y,\varepsilon) &= y + \varepsilon\left[R_\theta\left(\cos(\tau),\sin(\tau),y,\varepsilon\right) - R_\theta\left(1,0,y,\varepsilon\right)\right] \\ \phi_{\theta,-}(\tau,y,\varepsilon) &= y + \varepsilon\left[R_{\theta,-}\left(\cos(\tau),\sin(\tau),y,\varepsilon\right) - R_{\theta,-}\left(1,0,y,\varepsilon\right)\right], \end{aligned} \tag{92}$$

où les perturbations $R_{\theta,F}$, R_θ et $R_{\theta,-}$ sont des perceptrons multicouche.

Pour créer des données, on prend K données initiales y_0^k au temps t_0^k, respectivement pris aléatoirement dans $\Omega \subset \mathbb{R}^d$ (supposé compact) et $[0, 2\pi]$. Dans un second temps, pour tout $k \in [\![0, K_1]\!]$, on calcule, pour ε^k et h^k, respectivement pris aléatoirement dans $[\varepsilon_-, \varepsilon_+]$ et $[h_-, h_+]$, le flot exact en $t_0^k + h^k$ (on utilise en réalité

une méthode précise, mais coûteuse car non adaptée aux problèmes non raides)

$$y_1^k = \varphi^f_{t_0^k,h^k}\left(y_0^k\right). \tag{93}$$

Une fois ces données créées, on peut entraîner les trois réseaux de neurones en minimisant la fonction *Loss* (erreur quadratique moyenne) sur K_0 premières données, mais qui va aussi entraîner l'auto-encodeur de telle sorte à ce que la paire de réseaux de neurones soit réciproque l'une à l'autre.

$$\begin{aligned} Loss_{Train} &= \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|y_1^k - \hat{y_1}^k\right|^2 \\ &+ \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|\phi_\theta\left(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\right)\circ\phi_{\theta,-}\left(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\right)\left(y_0^k\right) - y_0^k\right|^2 \\ &+ \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|\phi_{\theta,-}\left(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\right)\circ\phi_\theta\left(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\right)\left(y_0^k\right) - y_0^k\right|^2, \end{aligned} \tag{94}$$

où, pour tout $k \in [\![0, K-1]\!]$, $\hat{y_1}^k$ est donné par

$$\hat{y_1}^k = \phi_\theta\left(\frac{t_0^k + h^k}{\varepsilon^k}, \Phi_{h^k}^{F_\theta(\cdot,h^k,\varepsilon^k)}\left(\phi_{\theta,-}\left(\frac{t_0^k}{\varepsilon^k}, y_0^k, \varepsilon^k\right)\right), \varepsilon^k\right). \tag{95}$$

Pour détecter un éventuel sur-apprentissage, on observe également la décroissance de la fonction $Loss_{Test}$, erreur quadratique moyenne sur les données restantes

$$\begin{aligned} Loss_{Test} &= \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\left|y_1^k - \hat{y_1}^k\right|^2 \\ &+ \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\left|\phi_\theta\left(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\right)\circ\phi_{\theta,-}\left(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\right)\left(y_0^k\right) - y_0^k\right|^2 \\ &+ \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\left|\phi_{\theta,-}\left(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\right)\circ\phi_\theta\left(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\right)\left(y_0^k\right) - y_0^k\right|^2. \end{aligned} \tag{96}$$

Une fois l'entraînement terminé, on dispose de $\phi_{\theta,+}$ et F_θ, respectivement approximations du générateur de fortes oscillations et du champ moyenné modifié. Ainsi, en introduisant la suite $(v_{\theta,n})_{n\in\mathbb{N}}$ définie de manière récursive par

$$\left\{\begin{array}{rcl} v_{\theta,0} & = & y^{\varepsilon}(0) \\ \forall n \in \mathbb{N}, \quad v_{\theta,n+1} & = & \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(v_{\theta,n}) \end{array}\right. , \tag{97}$$

ainsi que la suite $(y_{\theta,n})_{n\in\mathbb{N}}$ définie pour tout $n \in \mathbb{N}$ par

$$y^{\varepsilon}_{\theta,n} = \phi_\theta\left(\frac{nh}{\varepsilon}, v_{\theta,n}, \varepsilon\right), \tag{98}$$

on obtient, en introduisant les erreurs d'apprentissage suivantes:

$$\delta_\phi := \left\| \frac{\phi^{\varepsilon,[n_\varepsilon]} - \phi_\theta}{\varepsilon} \right\|_{L^\infty([0,2\pi]\times\Omega\times[0,\varepsilon_+])} \tag{99}$$

$$\delta_F := \left\| \widetilde{F_h^{\varepsilon,[n_\varepsilon],[q]}} - F_\theta \right\|_{L^\infty(\Omega\times[0,h_+]\times[0,\varepsilon_+])} , \tag{100}$$

où $\widetilde{F_h^{\varepsilon,[n_\varepsilon],[q]}}$ est le champ modifié $\widetilde{F_h^{\varepsilon,[n_\varepsilon]}}$ tronqué à l'ordre q, l'estimation d'erreur suivante (N est le nombre d'itérations et $h = \frac{T}{N}$) est de la forme:

$$\underset{0\leqslant n\leqslant N}{Max} \left| y^{\varepsilon}(t_n) - y^{\varepsilon}_{\theta,n} \right| \leqslant M e^{-\frac{\beta\varepsilon_0}{\varepsilon}} + \delta_\phi \cdot \varepsilon + C\delta_F, \tag{101}$$

où $C > 0$ est indépendante de h et ε et p est l'ordre de la méthode numérique Φ_h.

Nous obtenons ainsi une borne intéressante par rapport à ε lorsque ce paramètre tend vers 0, mais plus grossière lorsque ce dernier tend vers 1, en raison du reste exponentiel. Par ailleurs, des tests numériques ont été menés sur l'oscillateur de Van der Pol ainsi que le Pendule inversé.

3.3.2 Ajout d'une correction Micro-Macro

De plus, on peut ajouter un terme correctif en s'inspirant de la méthode Micro-Macro. En effet, considérons maintenant la suite $(w_{\theta,n})_{n\in\mathbb{N}}$ donnée, par:

$$\left\{\begin{array}{rcl} w_{\theta,0} & = & 0 \\ \forall n \in \mathbb{N}, \quad w_{\theta,n+1} & = & \left(\Phi_{t_n,h}^{g_\theta(\frac{t_n}{\varepsilon},\cdot,v_{\theta,n},\varepsilon)}\right)(w_{\theta,n}) \end{array}\right. \tag{102}$$

où, pour tout $(\tau, w, v, \varepsilon) \in [0, 2\pi] \times \Omega \times \Omega\times]0, 1]$,

$$g_\theta(\tau, w, v, \varepsilon) = f(\tau, \phi_\theta(\tau, v, \varepsilon) + w) - \frac{1}{\varepsilon}\partial_\tau \phi_\theta(\tau, v, \varepsilon) - \partial_y \phi_\theta(\tau, v, \varepsilon) F_\theta(v, 0, \varepsilon), \tag{103}$$

ainsi que, cette fois-ci, la suite $(y^\varepsilon_{\theta,n})_{n\in\mathbb{N}}$ donnée, pour tout $n \in \mathbb{N}$, par:

$$y^\varepsilon_{\theta,n} = \phi_\theta\left(\frac{nh}{\varepsilon}, v_{\theta,n}, \varepsilon\right) + w_{\theta,n}, \tag{104}$$

alors on obtient, en introduisant les erreurs d'apprentissage suivantes:

$$\delta_\phi := \left|\left|\frac{\phi^{[p]} - \phi_\theta}{\varepsilon}\right|\right|_{W^{1,\infty}(\Omega\times[0,2\pi]),L^\infty([0,\varepsilon_+])} \tag{105}$$

$$\delta_F := \left|\left|\widetilde{F_h^{\varepsilon,[p],[q]}} - F_\theta\right|\right|_{L^\infty(\Omega\times[0,h_+]\times[0,\varepsilon_+])}, \tag{106}$$

une estimation d'erreur de la forme:

$$\underset{0\leqslant n\leqslant N}{Max}\left|y^\varepsilon(t_n) - y^\varepsilon_{\theta,n}\right| \leqslant M' h^p + C'\left[\delta_F + \delta_\phi\right], \tag{107}$$

où $M', C' > 0$ sont indépendantes de h et ε et p est l'ordre de la méthode numérique $\Phi_{\cdot\cdot}$.

Nous obtenons ainsi une borne intéressante, qui est uniforme par rapport à ε (le problème lié au reste exponentiel $Me^{-\frac{\beta\varepsilon_0}{\varepsilon}}$, présent sans cette correction Micro-Macro est donc corrigé). Par ailleurs, des tests numériques ont été menés sur l'oscillateur de Van der Pol ainsi que le Pendule inversé.

3.3.3 Cas autonome fortement oscillant

Ce cas particulier s'intéresse au problème suivant:

$$\dot{y^\varepsilon}(t) = \frac{1}{\varepsilon} A y^\varepsilon(t) + g(y^\varepsilon(t)) \tag{108}$$

où A est une matrice dont les valeurs propres sont dans $i\mathbb{Z}$ et $g : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ est une fonction régulière, à priori non linéaire. On sait qu'il existe A^ε et g^ε tels que A^ε génère un flot $2\pi-$périodique $\tau \longmapsto \phi^\varepsilon_\tau = e^{\tau A^\varepsilon}$ (dynamique rapide), g^ε génère un

flot $t \longmapsto \varphi_t^{g^\varepsilon}$ (dynamique lente), que ces deux flots commutent et que, pour tout $\varepsilon \in]0,1]$:

$$\left| y^\varepsilon(t) - \phi^\varepsilon_{\frac{t}{\varepsilon}}\left(\varphi_t^{g^\varepsilon}(y_0)\right)\right| \leqslant C_T e^{-\frac{C_T}{\varepsilon}}, \tag{109}$$

où $C_T > 0$ est une constante indépendante de ε.

Dans ce cas de figure, A^ε et g^ε sont difficilement calculables, et on propose ici d'apprendre les deux flots ϕ^ε et φ^{g^ε} via des réseaux de neurones, notés respectivement ϕ_θ et φ_θ et possédant cette structure:

$$\begin{aligned}
\varphi_\theta(y,h,\varepsilon) &= y + hR_{\theta,\varphi}(y,h,\varepsilon) &(110)\\
\phi_\theta(\tau,y,\varepsilon) &= y + \varepsilon\Big[R_{\theta,\phi}(\cos(\tau),\sin(\tau),y,\varepsilon) - R_{\theta,\phi}(1,0,y,\varepsilon)\Big], &(111)
\end{aligned}$$

où les perturbations $R_{\theta,\varphi}$ et $R_{\theta,\phi}$ sont toujours des perceptrons multicouche.

Pour créer des données, on se donne K conditions initiales y_0^k choisies aléatoirement dans $\Omega \subset \mathbb{R}^d$ (supposé compact), et, pour tout $k \in [\![0, K-1]\!]$, on choisit aléatoirement $h^k \in [h_-, h_+]$ et $\varepsilon^k \in [\varepsilon_-, \varepsilon_+]$ puis on calcule le flot exact (encore une fois avec une méthode précise mais coûteuse car non adaptée aux problèmes raides) en h^k, noté $y_1^k = y^{\varepsilon^k}\left(h^k\right)$.

On entraîne les réseaux de neurones en minimisant la fonction *Loss*, erreur quadratique moyenne sur K_0 données:

$$\begin{aligned}
Loss_{Train} &= \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left| y_1^k - \phi_\theta\left(\frac{h^k}{\varepsilon^k}, \varphi_\theta(y_0^k, h^k, \varepsilon^k), \varepsilon^k\right)\right|^2 \\
&+ \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|\varphi_\theta\left(\phi_\theta\left(\frac{h^k}{\varepsilon^k}, y_0^k, \varepsilon^k\right), h^k, \varepsilon^k\right)\right. \\
&\quad \left. -\phi_\theta\left(\frac{h^k}{\varepsilon^k}, \varphi_\theta(y_0^k, h^k, \varepsilon^k), \varepsilon^k\right)\right|^2,
\end{aligned} \tag{112}$$

cette fois-ci, le fait que les flots commutent n'impose pas l'utilisation d'un auto-encodeur pour apprendre ϕ. Toutefois, on force la commutativité en ajoutant un

terme dans la fonction *Loss*, qui va faire commuter "au mieux" les deux réseaux de neurones.

On regarde également la fonction $Loss_{Test}$ afin d'éviter le sur-apprentissage.

$$\begin{aligned} Loss_{Test} &= \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\left|y_1^k-\phi_\theta\left(\frac{h^k}{\varepsilon^k},\varphi_\theta(y_0^k,h^k,\varepsilon^k),\varepsilon^k\right)\right|^2 \\ &+ \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\left|\varphi_\theta\left(\phi_\theta\left(\frac{h^k}{\varepsilon^k},y_0^k,\varepsilon^k\right),h^k,\varepsilon^k\right)\right. \\ &\quad \left.-\phi_\theta\left(\frac{h^k}{\varepsilon^k},\varphi_\theta(y_0^k,h^k,\varepsilon^k),\varepsilon^k\right)\right|^2. \end{aligned} \tag{113}$$

Une fois l'entraînement terminé, on dispose de ϕ_θ et φ_θ, correspondant aux deux flots appris. Soit $(y^\varepsilon_{\theta,n})_{n\in\mathbb{N}}$ la suite définie, pour tout $n\in\mathbb{N}$, par:

$$y^\varepsilon_{\theta,n}=\phi_\theta\left(\frac{nh}{\varepsilon},\varphi_\theta(\cdot,h,\varepsilon)^n(y^\varepsilon(0))\right). \tag{114}$$

En introduisant les erreurs d'apprentissage suivantes:

$$\delta_\phi := ||\phi^\varepsilon-\phi_\theta||_{L^\infty([0,2\pi]\times\Omega\times[0,\varepsilon_+])} \text{ et } \delta_\varphi := \left|\left|\frac{\varphi^{g^\varepsilon}-\varphi_\theta}{h}\right|\right|_{L^\infty(\Omega\times[0,h_+]\times[0,\varepsilon_+])}, \tag{115}$$

on obtient ainsi:

$$\underset{0\leqslant n\leqslant N}{Max}\left|y^\varepsilon_{\theta,n}-y^\varepsilon(t_n)\right|\leqslant\Lambda(\delta_\phi+\delta_\varphi)+C_Te^{-\frac{C_T}{\varepsilon}}. \tag{116}$$

où $\Lambda>0$ est une constante indépendante de ε.

Nous obtenons ainsi une méthode ne nécessitant pas d'auto-encodeur (donc plus simple du point de vue de la structure de réseau de neurones) et permettant une intégration numérique efficace de ce type d'équation différentielle. De plus, des tests numériques ont été menés sur le système de Van der Pol.

3.4 Chapitre 4: PINN's pour le cas fortement oscillant

Dans ce dernier chapitre, nous allons également exploiter la structure multi-échelle des solutions d'équations fortement oscillantes afin d'obtenir une résolution numérique via les réseaux de neurones.

Toutefois, nous utiliserons ici un PINN, qui inclut directement l'équation et la condition initiale dans la fonction *Loss*. Soit $\varepsilon \in]0,1]$ fixé. En notant φ_θ^f le réseau de neurones chargé d'apprendre $(t,y) \longmapsto \varphi_t^f(y)$, la solution exacte de condition initiale y, on minimise la fonction *Loss* donnant l'erreur quadratique moyenne sur l'équation ainsi que la condition initiale:

$$\begin{aligned} Loss_{Train} &= \frac{\xi}{K_0}\sum_{k=0}^{K_0-1}\left|\frac{\partial\varphi_\theta^f}{\partial t}(t_k,x_k) - f\left(\frac{t_k}{\varepsilon},\varphi_\theta^f(t_k,x_k)\right)\right|^2 \\ &+ \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|\varphi_\theta^f(0,x_k)-x_k\right|^2 \end{aligned} \tag{117}$$

où $\xi > 0$ est un coefficient donnant plus ou moins d'importance à l'apprentissage de l'équation ou de la condition intiale. On remarque que le premier terme va forcer le réseau de neurones à vérifier l'équation tandis que le deuxième va le forcer à vérifier la condition initiale. Les points x_k et t_k sont choisis aléatoirement, respectivement sur $\Omega \subset \mathbb{R}^d$ (supposé compact) et $[0,T]$, l'intervalle de résolution.

Pour éviter le sur-apprentissage, on regarde également l'évolution de la fonction $Loss_{Test}$ en les valeurs x_k et t_k restantes:

$$\begin{aligned} Loss_{Test} &= \frac{\xi}{K-K_0}\sum_{k=K_0}^{K-1}\left|\frac{\partial\varphi_\theta^f}{\partial t}(t_k,x_k) - f\left(\frac{t_k}{\varepsilon},\varphi_\theta^f(t_k,x_k)\right)\right|^2 \\ &+ \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\left|\varphi_\theta^f(0,x_k)-x_k\right|^2. \end{aligned} \tag{118}$$

De plus, φ_θ^f possède une structure sous-jacente suivant la décomposition en deux dynamiques (lente et rapide) introduite au chapitre précédent:

$$\varphi_\theta^f(t,y) = \phi_\theta\left(\frac{t}{\varepsilon}, \psi_\theta(t,y)\right), \tag{119}$$

où ψ_θ est un perceptron multicouche et ϕ possède la structure suivante:

$$\phi(\tau,y) = y + \varepsilon\left[R_\theta^\phi(\cos(\tau),\sin(\tau),y) - R_\theta^\phi(1,0,y)\right], \tag{120}$$

où la perturbation R^ϕ est également un perceptron multicouche.

Ainsi, en notant les erreur d'apprentissage suivantes:

$$\delta_{ODE} := \underset{(t,x)\in[0,T]\times\Omega}{Max}\left|\frac{\partial\varphi_\theta^f}{\partial t}(t,x) - f\left(\frac{t}{\varepsilon},\varphi_\theta^f(t,x)\right)\right| \text{ et } \delta_{IC} := \underset{x\in\Omega}{Max}\left|\varphi_\theta^f(0,x) - x\right|, \tag{121}$$

respectivement sur l'équation et la condition initiale, alors on a l'estimation d'erreur suivante:

$$\underset{(t,x)\in[0,T]\times\Omega}{Max}\left|\varphi_\theta^f(t,x) - \varphi_{0,t}^f(x)\right| \leqslant (\delta_{IC} + T\delta_{ODE})\, e^{\tilde{L}_f T} \tag{122}$$

où $\tilde{L}_f$ est une constante indépendante de T.

Cette méthode, très courante pour la résolution d'EDP's par réseaux de neurones, permet ainsi d'obtenir une approximation de la solution sans utiliser de données créées au préalable. De plus, des tests numériques ont été menés sur l'oscillateur de Van der Pol ainsi que sur le système Hénon-Heiles, qui est Hamiltonien, avec une étude de la conservation de l'Hamiltonien au cours du temps.

4 Perspectives de recherche

Voici quelques pistes pouvant être explorées pour de futures recherches.

4.1 Propriétés géométriques

Il est possible de s'intéresser à des équations fortement oscillantes hamiltoniennes, c'est-à-dire de la forme

$$\dot{y^{\varepsilon}}(t) = J^{-1}\nabla_y H\left(\frac{t}{\varepsilon}, y^{\varepsilon}(t)\right) \tag{123}$$

où la fonction H, appelée hamiltonien, est supposée régulière et périodique en sa première variable. Si l'on souhaite étudier la décomposition en deux dynamiques $t \longmapsto \phi_{\frac{t}{\varepsilon}}^{[n]}(\varphi_t^{F^{[n]}}(y^{\varepsilon}(0)))$, cette dernière existe toujours à un reste exponentiel près, avec en plus comme propriété que $F^{[n]}$ est un champ de vecteurs hamiltonien, et que l'application $\phi_\tau^{[n]}$ est symplectique.

Ainsi, il peut être envisageable d'apprendre ces deux applications avec des réseaux de neurones en y incluant les propriétés géométriques, c'est-à-dire que l'on peut apprendre $F^{[n]}$ sous forme d'un champ de vecteurs Hamiltonien, et ce au moyen d'un HNN via l'utilisation d'une méthode symplectique [32]:

$$F_\theta(., h, \varepsilon) = J^{-1}\nabla_y H_\theta(\cdot, h, \varepsilon). \tag{124}$$

En effet, le champ modifié d'un champ hamiltonien associé à une méthode symplectique est également hamiltonien [50], et ici H_θ est un réseau de neurones à valeurs dans $\mathbb{R}$.

De plus, une stratégie permettant d'obtenir une approximation du caractère symplectique peut être d'ajouter les termes suivants à la *Loss*, termes portant sur $\phi_{\theta,+}$ et $\phi_{\theta,-}$:

$$\frac{1}{K_0}\sum_{k=0}^{K_0-1}\left\|\frac{\partial\phi_\theta}{\partial y}\left(\frac{t_0^k}{\varepsilon^k}, y_0^k, \varepsilon^k\right)^T J\frac{\partial\phi_\theta}{\partial y}\left(\frac{t_0^k}{\varepsilon^k}, y_0^k, \varepsilon^k\right) - J\right\|^2, \tag{125}$$

$$\frac{1}{K_0}\sum_{k=0}^{K_0-1}\left\|\frac{\partial\phi_{\theta,-}}{\partial y}\left(\frac{t_0^k}{\varepsilon^k}, y_0^k, \varepsilon^k\right)^T J\frac{\partial\phi_{\theta,-}}{\partial y}\left(\frac{t_0^k}{\varepsilon^k}, y_0^k, \varepsilon^k\right) - J\right\|^2. \tag{126}$$

puisque la réciproque d'une application symplectique l'est également. De cette manière, il peut être envisageable d'obtenir une solution préservant mieux l'Hamiltonien.

Une variante possible concerne le cas où $div_y(f) = 0$ (champ à divergence nulle). Dans ce cas, le champ moyenné est également à divergence nulle, et l'application $\phi_\tau^{[n]}$ préserve les volumes, i.e. sa différentielle en espace est de déterminant 1. On peut

donc ajouter à la fonction *Loss* les termes suivants

$$\frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|\det\left(\frac{\partial\phi_{\theta,-}}{\partial y}\left(\frac{t_0^k}{\varepsilon^k},y_0^k,\varepsilon^k\right)\right)-1\right|^2+\frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|\det\left(\frac{\partial\phi_{\theta}}{\partial y}\left(\frac{t_0^k}{\varepsilon^k},y_0^k,\varepsilon^k\right)\right)-1\right|^2 \tag{127}$$

et

$$\frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|div_y\left(F_\theta(y_0^k,0,\varepsilon^k\right)\right|^2, \tag{128}$$

respectivement pour inclure dans l'apprentissage le fait que le changement de variable (et donc sa réciproque) préservent les volumes et que le champ moyenné est à divergence nulle. Pour le champ moyenné, une alternative peut être d'utiliser des VP-Nets.

4.2 Dépendance en ε

Dans le cas de l'amélioration du schéma d'Euler intégral ainsi que dans celui des PINN's, le paramètre de forte oscillation est fixé, or, il peut être intéressant de le faire varier.

Pour le schéma d'Euler intégral, où une méthode UA améliorée peut ainsi être créée, et ce sans avoir à refaire un apprentissage à chaque changement de paramètre ε. Une stratégie possible peut être d'augmenter le nombre d'entrées du réseau de neurones R_θ, en y ajoutant des dépendances en ε, $\cos\left(\frac{h}{\varepsilon}\right)$ et $\sin\left(\frac{h}{\varepsilon}\right)$.

$$\begin{aligned}\hat{y_1}^k &= y_0^k+\int_{t_0^k}^{t_0^k+h^k} f\left(\frac{t}{\varepsilon},y_0^k\right)\mathrm{d}t \\ &+ h^2R_\theta\left(\cos\left(\frac{t_0^k}{\varepsilon^k}\right),\sin\left(\frac{t_0^k}{\varepsilon^k}\right),y_0^k,h^k,\varepsilon^k,\cos\left(\frac{h^k}{\varepsilon^k}\right),\sin\left(\frac{h^k}{\varepsilon^k}\right)\right).\end{aligned} \tag{129}$$

Ainsi, un seul apprentissage (comprenant toutefois plus de données) serait nécessaire.

Pour le cas des PINN's, une stratégie envisageable pourrait également d'augmenter le nombre d'entrées des réseaux de neurones ϕ_θ et ψ_θ apprenant les deux dynamiques

$$\phi_\tau^\varepsilon(y) \approx \phi_\theta(\tau, y, \varepsilon) \tag{130}$$

$$\varphi_t^{F^{[n]},\varepsilon}(y) \approx \psi_\theta(t, y, \varepsilon) \tag{131}$$

$$\varphi_\theta^f(t, y, \varepsilon) \approx \phi_\theta\left(\frac{t}{\varepsilon}, \psi_\theta(t, y, \varepsilon), \varepsilon\right) \tag{132}$$

$$\begin{aligned} Loss_{Train} &= \frac{\xi}{K_0}\sum_{k=0}^{K_0-1}\left|\frac{\partial\varphi_\theta^f}{\partial t}(t_k, x_k, \varepsilon_k) - f\left(\frac{t_k}{\varepsilon_k}, \varphi_\theta^f(t_k, x_k, \varepsilon_k)\right)\right|^2 \\ &+ \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|\varphi_\theta^f(0, x_k, \varepsilon_k) - x_k\right|^2. \end{aligned} \tag{133}$$

De plus, la structure devrait peut-être être modifiée afin de tenir compte des paramètres ε proches de 1.

5 Articles de recherche

La thèse a donné lieu à la rédaction de deux articles de recherche:

- Maxime Bouchereau, Philippe Chartier, Mohammed Lemou & Florian Méhats, *Machine Learning Methods for Autonomous Ordinary Differential Equations*, 2023, accepté dans *Communications in Mathematical Sciences* en Décembre 2023.
- Maxime Bouchereau, Philippe Chartier, Mohammed Lemou & Florian Méhats, *Machine Learning Methods for Highly Oscillatory Differential Equations*, 2024, à soumettre.

Thesis work

Chapter 1

Machine Learning methods for Autonomous differential equations

This chapter is based on the paper [10].

Abstract

Ordinary Differential Equations are generally too complex to be solved analytically. Approximations thereof can be obtained by general purpose numerical methods. However, even though accurate schemes have been developed, they remain computationally expensive: In this paper, we resort to the theory of modified equations in order to obtain "on the fly" cheap numerical approximations. The recipe consists in approximating, prior to that, the modified field associated to the modified equation by *neural networks*. Elementary convergence results are then established and the efficiency of the technique is demonstrated on experiments.

Keywords: modified equation, ordinary differential equation, neural network, numerical method, convergence analysis.

1.1 Introduction

Ordinary Differential Equations are ubiquitous in the modelling of systems in domains of science as diverse as biology, dynamics, fluid mechanics, quantum mechanics, thermodynamics or weather forecasting. In most situations, these differen-

tial equations can not be solved analytically and necessitate the use of numerical methods so as to compute accurate approximations [48, 50].

Generally speaking, the aforementioned methods are the result of a trade-off between accuracy and computational cost. In simple words, a small approximation error requires long computations. Of course, there are various ways in which one may soften the computational constraints, e.g. by raising the order, taking into account the structure of the problem or optimising the coefficients of the method. Excellent numerical methods are abundant in the literature and we refer to reference books [8, 12, 15, 48, 50, 66] for them.

In this paper, we choose to derive as much as possible prior information from the knowledge of the vector field of the equation in order to accelerate the solving of the equation. This is thus a two-step process: (i) first, generate data that will be used during the effective computation of the approximate solution; (ii) second, compute, from an initial value, an approximation of the solution "on the fly" as accurately as possible by using available pre-computed values. As such, the process remains too vague to become practical: this is where the crucial ingredient of the technique comes into play, that is to say modified equations.

The theory of modified equations has emerged in the context of ordinary differential equations as a powerful tool (referred to as *backward error analysis*) to explain the excellent behaviour of structure-preserving numerical methods [50]. More recently, modified fields have also been used in a dual manner, as a technique to raise the order of any existing numerical method by twisting appropriately the original vector field [8]. The idea is that, by adding ad-hoc perturbation terms to the vector field and then solving the associated differential equation, one may compute higher-order approximations. It can be indeed proved that, at least at a formal level, for any numerical method, there exists a modified equation whose solution by the aforementioned method coincides with the exact solution. The perturbation terms to be added may be obtained in analytical form as *elementary differentials* involving various derivatives of the original vector field. Complete expansions are for instance available with the help of representations by trees (known as B-series). However, computing the corresponding expressions analytically would be an extremely tedious process and this is the reason why the technique has remained confined to specific situations [20, 50, 51] of limited practical interest.

The main idea of this work is thus to combine the theory of modified equa-

tions with machine learning techniques and more precisely neural networks. Given a differential equation and a numerical method, the ad-hoc perturbations of the vector field are first *learnt* by extensive simulations and then approximated by inference from a neural network. Once this representation of the modified vector field has been obtained, it is used to solve the original equation with the same numerical method for any initial value prescribed in a *learnt* domain from the phase-space. The combined computational work ids by far greater than for any usual reasonable numerical method. Nevertheless, if one omits the time spend to learn the perturbation, an accurate solution can be obtained very cheaply as compared to well-established schemes such as those of Dormand & Prince [90].

1.1.1 Scope of the chapter

The article is divided into two main sections: Section 1.2 is devoted to the exposition of the technique and its convergence analysis, while Section 1.3 presents numerical experiments illustrating the performances and properties of the schemes we analysed.

Subsection 1.2.1 exposes the general strategy which is adopted. A specific structure of the neural network is selected, in agreement with the structure of the modified field. Moreover, the details of the machine leaning method for the learning of the modified field are given here, concerning in particular the choice of the training data set and the calibration of the parameters of the neural network (approximating the modified field via *loss*-minimization).

Subsection 1.2.2.1 establishes a convergence result for the numerical integration resulting from the combined use of neural networks and classical schemes. The essential difference with standard convergence results is reflected in the multiplicative constant of the local error which turns out to be smaller than for a direct application to the original vector field. A special focus is put on explicit Runge-Kutta methods in the same subsection, where more precise estimates are given.

Moreover, two specific schemes are studied in this subsection: the forward Euler method and a Runge-Kutta method of order 2, which are two simple occurrences of explicit methods. A short convergence analysis is undertaken for each of them, leading to improved bounds. The same methods are then used for numerical experiments which illustrate two main contributions of this work: (i) The modified field can be learnt efficiently through a neural network, as is illustrated in 1.3.1;

(ii) The resulting numerical methods have far greater efficiency, as is illustrated on convergence curves in Subsection 1.3.2. A comparison with the well-known DOPRI5 method in Subsection 1.3.3 is particularly enlightening with this respect.

1.1.2 Related work

The link between differential equations and machine learning has been already explored in several publications. Two approaches are prominent. On the one hand, the ODE vector field can be learnt by the technique of MSE Loss, which can be applied to both ODEs or PDEs [93]. Let us notice also a paper of Burton et al. [11] which proposes symbolic regression for the learning of complex dynamical systems. On the other hand, the ODE vector field can be learnt by statistical methods. for instance, Expectation-Maximization is used in a paper of Nguyen et al. [81], while a paper of Raissi et al. [92] proposes Gaussian processes for linear differential equations.

Links with modified equations. Links between machine learning and modified equations have been established more recently, e.g. in the paper [118] where the theory of modified equation is used fora rigorous analysis. Offen et al. [85] consider numerical methods for Hamiltonian ODEs. The methods used therein are statistical methods, namely Gaussian processes.

Structure of Neural Networks. In order to preserve geometric properties, specific neural networks, adapted to the structure of the differential equations under consideration have been developed. A first example are Hamiltonian equations for which Hamiltonian neural networks have been developed (see [32] or [59] where irregular time observed data can be used). A second example are Poisson systems for which Jin et al. [58] developed Poisson neural networks. Another example of specific ODEs is studied in another recent paper [119], where VPNets are used to learn the volume-preserving flows.

ODEs methods for Neural Networks. In the same way as neural networks are used to solve ODEs, the reciprocal strategy can be pursued: neural networks can indeed be modelled by ODEs and their properties deduced from the corresponding ODE properties. For instance, Lu et al. [73] have developed new neural networks which are discretizations of ODEs by various numerical methods and Haber et al. [46] have developed new structures of neural networks depending on the stability properties of the ODEs. Finally, Chen et al. [27] have derived new optimization

methods of the loss function from the properties of the corresponding ODEs (the neural network is here again obtained through the discretization of the ODE).

Approximation by neural networks. In order to properly approximate functions by neural networks, error estimates have been established. Anastassiou [1] stated rates of convergence for approximations of functions by networks, according to the number of parameters and the dimension. By considering neural network spaces as functional spaces, Gribonval et al. [45] have obtained inclusions of theses spaces in Besov or Lebesgue spaces, according to the number of parameters and a given rate of convergence. In a separate work, Bach [4] has given bounds on the approximation error in a Hilbertian setting. Finally, a problem of approximation [14] which is underlined is the curse of dimensionality, where high-dimensional vector fields are approximated with a slower rate of convergence than low-dimensional vector fields, i.e. for high dimensions, more parameters and more data will be required in order to get a satisfying learning.

1.2 Improving the accuracy of numerical methods with machine learning

Consider an autonomous ordinary differential equation of the form

$$\left\{ \begin{array}{rcl} \dot{y}(t) & = & f(y(t)) \in \mathbb{R}^d, \quad t \in [0,T] \\ y(0) & = & y_0 \end{array} \right. ,$$

where $f : \mathbb{R}^d \to \mathbb{R}^d$ is assumed to be smooth enough. By Cauchy-Lipschitz theorem, we have existence and uniqueness of a solution for any given initial value $y_0 \in \mathbb{R}^d$. We wish to approximate the solution over $[0,T]$ at times $t_n = nh$, $0 \leqslant n \leqslant N$, where $h = \frac{T}{N}$ is the time-step and N is the number of discretization points.

1.2.1 General strategy

As explained in the Introduction section, we shall approximate the modified vector field with the help of a neural network.

1.2.1.1 Modified field

Let us consider $\Phi_h^f(\cdot)$ the numerical flow associated to a given numerical method (h is the time-step of the method) and to the vector field f, and assume that it is of order p, in the sense that [1]

$$\underset{0\leqslant n\leqslant N}{Max}\left|\left(\Phi_h^f\right)^n(y_0)-\varphi_{nh}^f(y_0)\right| \;\leqslant\; Ch^p \tag{1.1}$$

for some constants $C>0$. If we modify the field f used in Φ_h, i.e. if we apply the numerical flow Φ_h with the modified field $\tilde{f}_h$ instead of f, we may obtain a higher-order approximation. In fact, the theory of *modified equations* states that it is possible to construct $\tilde{f}_h$ as a series of powers of h multiplied by appropriately chosen functions (at least as a formal series), in such a way that $(\Phi_h^{\tilde{f}_h})^n(y_0)$ coincides exactly with $y(t_n)$. The structure of this modified field writes (see [50])

$$\tilde{f}_h(y)=f(y)+h^p\sum_{j=1}^{+\infty}h^{j-1}f^{[j]}(y)=f(y)+h^p\sum_{j=1}^{k-1}h^{j-1}f^{[j]}(y)+h^{k+p-1}R(y,h) \tag{1.2}$$

where the coefficient-functions $f^{[j]}$ are built upon derivatives of f. It can be shown rigorously that the truncation of this formal series (1.2) obtained by neglecting the $\mathcal{O}\left(h^{k+p-1}\right)$-terms, leads to

$$(\Phi_h^{\tilde{f}_h})^n(y_0)=\varphi_{nh}^f(y_0)+\mathcal{O}\left(h^{k+p-1}\right)$$

where k denotes the number of terms kept in $\tilde{f}_h$.

1.2.1.2 Machine learning methods

The main idea of this paper consists in approximating the modified field $\tilde{f}_h$ by a neural network approximation $f_\theta(\cdot,h)$ whose structure mimics the structure of the theoretical modified field (1.2). More precisely, we shall approximate separately each function $f^{[j]}$ in (1.2) with a neural network. As could be anticipated, the truncation of (1.2) will be echoed by a similar truncation

1. Here and in the sequel, $|\cdot|$ denotes a norm on $\mathbb{R}^d$.

$$f_\theta(y,h) = f(y) + h^p \sum_{j=1}^{N_t-1} h^{j-1} f_j(y) + h^{N_t+p-1} R_a(y,h), \tag{1.3}$$

where f_i, for $1 \leqslant i \leqslant N_t - 1$, and R_a, are Multi Layer Perceptrons. An obvious advantage of this choice is that the corresponding numerical method remains consistent. Let us further notice that learning the perturbation in this way is also well-adapted to the situations where the original vector field possesses a specific structure [32, 58, 59, 93, 119].

The complete numerical procedure can be decomposed into three main steps : Firstly, data are collected by simulating very accurately the exact flow at various points of the domain. A high number of simulations and a high accuracy are prerequisite for a good approximation of the modified field. Secondly, the different neural networks are trained separately by minimising a prescribed *Loss*-function. Eventually, given an initial data y_0, an approximation of the exact solution is obtained by applying the same numerical scheme as the one used for the training to the field f_θ. In more details, we follow the three stages:

1. **Construction of the data set:** K initial data $y_0^{(k)}$ are randomly selected into a compact set $\Omega \subset \mathbb{R}^d$ (where we wish to simulate the solution) with uniform distribution. Then, for all $0 \leqslant k \leqslant K-1$, we compute a very accurate approximation of the the exact flow at times $h^{(k)}$ with initial condition $y_0^{(k)}$, denoted $y_1^{(k)}$. Time steps $h^{(k)}$ are selected in the domain $[h_-, h_+]$ (we actually pick up the value $\log h^{(k)}$ randomly in the domain $[\log h_-, \log h_+]$ with uniform distribution).

2. **Training of the neural networks:** We minimize the Mean Squared Error (MSE), denoted $Loss_{Train}$, which measures the difference between predicted data $\hat{y}_1^{(k,\ell)}$ and "exact data" $y_1^{(k,\ell)}$ by computing the optimal NN's parameters over K_0 data (where $1 \leqslant K_0 \leqslant K-1$) by resorting to a gradient method:

$$Loss_{Train} = \frac{1}{K_0} \sum_{k=0}^{K_0-1} \frac{1}{h^{(k)2p+2}} \left| \underbrace{\Phi_{h^{(k)}}^{f_\theta(\cdot,h^{(k)})}\left(y_0^{(k)}\right)}_{=\hat{y_1}^{(k)}} - \underbrace{\varphi_{h^{(k)}}^{f}\left(y_0^{(k)}\right)}_{=y_1^{(k)}} \right|^2 \tag{1.4}$$

At the same time, we compute the value of another MSE, denoted $Loss_{Test}$, which measures the difference between predicted data $\hat{y}_1^{(k,\ell)}$ and "exact data" $y_1^{(k,\ell)}$ for a subset of the initial values which have not been used to train the NNs. The objective of this step is to estimate the performance of the training for "unknown" initial values :

$$Loss_{Test} \quad = \quad \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\frac{1}{h^{(k)^{2p+2}}}\left|\Phi_{h^{(k)}}^{f_\theta(\cdot,h^{(k)})}\left(y_0^{(k)}\right)-\varphi_{h^{(k)}}^{f}\left(y_0^{(k)}\right)\right|^2 \quad (1.5)$$

If $Loss_{Train}$ and $Loss_{Test}$ exhibit the same decay pattern, one considers that there is no overfitting, that is to say that the neural network model does not fit exactly against its training data and remains able to perform accurately against unseen data, which is its main purpose.

3. **Numerical approximation:** At the end of the training, an accurate approximation $f_\theta(\cdot,h)$ of $\tilde{f}_h$ is available. It is then used to compute the successive values of $(\Phi_h^{f_\theta(\cdot,h)})^n(y_0)$ for $n=0,\ldots,N$.

1.2.2 Error analysis

1.2.2.1 General case

In this sub-subsection, we analyse the error resulting from the procedure described in previous Subsection. More specifically, we state estimates of the global error for any standard numerical method.

Theorem 24. *Let us denote $\Phi_h^{f_\theta(\cdot,h)}$, the flow of a given numerical scheme Φ_h of order p, applied to the modified field $f_\theta(\cdot,h)$ and let us consider the global error*

$$e_n \quad := \quad \left(\Phi_h^{f_\theta(\cdot,h)}\right)^n(y_0)-\left(\varphi_h^f\right)^n(y_0), \quad (1.6)$$

at times $t_n=nh$ for $0\leqslant n\leqslant N$. Denoting the learning error by

$$\delta \quad := \quad \max_{(y,h)\in\Omega\times[h_-,h_+]}\frac{\left|\tilde{f}_h(y,h)-f_\theta(y,h)\right|}{h^p} \quad (1.7)$$

and assuming that

(i) *For any pair smooth vector fields f_1 and f_2, we have*

$$\forall 0 \leq h \leq h_+, \quad \left|\left|\Phi_h^{f_1} - \Phi_h^{f_2}\right|\right|_{L^\infty(\Omega)} \leqslant Ch\,||f_1 - f_2||_{L^\infty(\Omega)} \tag{1.8}$$

for some positive constant C, independent of f_1 and f_2;

(ii) *For any smooth vector field f, there exists a constant $L > 0$ such that*

$$\forall 0 \leq h \leq h_+, \forall (y_1, y_2) \in \Omega^2, \quad \left|\Phi_h^f(y_1) - \Phi_h^f(y_2)\right| \leqslant (1 + Lh)\,|y_1 - y_2| \tag{1.9}$$

Then there exist two constants $\tilde{C}, \tilde{L} > 0$ such that:

$$\underset{0\leqslant n\leqslant N}{Max}\,|e_n| \leqslant \frac{C\delta h^p}{\tilde{L}}\left(e^{\tilde{L}T} - 1\right) \tag{1.10}$$

Remarks. ***(i)*** *The vector fields f and $\tilde{f}_h$ are smooth, respectively by assumption and by construction. As for $f_\theta(\cdot, h)$, it is smooth as well given that it is obtained through the composition of affine functions $A_1, \cdots, A_{L+1}$ and nonlinear functions $\Sigma_1, \cdots, \Sigma_L$ (the so-called activation functions). The output of the NN thus appears to be of the form $A_{L+1} \circ \Sigma_L \circ A_L \circ \cdots \Sigma_1 \circ A_1$ (for L layers). Hence, if the activation functions are smooth, then so is $f_\theta(\cdot, h)$. This is the case for instance if the Σ_i's are the hyperbolic tangent functions.*

(ii) *Assumption (1.8) is straightforwardly satisfied for all known consistent methods.*

(iii) *A similar error estimate holds for a variable step-size implementation of the numerical method Φ: if we indeed use the sequence of steps $0 \leq h_j \leq h_+$, then $T = h_0 + \cdots, +h_{N-1}$ and $y_{n+1}^* = \Phi_{h_n}^{f_\theta(\cdot,h_n)}(y_n^*)$, then there exist $\tilde{C}, \tilde{L} > 0$ such that:*

$$\underset{0\leqslant n\leqslant N}{Max}\left|\Phi_{h_{n-1}}^{f_\theta(\cdot,h_{n-1})} \circ \ldots \circ \Phi_{h_0}^{f_\theta(\cdot,h_0)}(y_0) - \varphi_{h_{n-1}}^f \circ \ldots \circ \varphi_{h_0}^f(y_0)\right| \leqslant \frac{\tilde{C}\delta h^p}{\tilde{L}}\left(e^{\tilde{L}T} - 1\right),$$

where $h = \underset{0 \leqslant j \leqslant N-1}{\text{Max}} h_j$.

1.2.2.2 Explicit Runge-Kutta method

In this other sub-subsection, we will focus over Explicit Runge-Kutta methods, which give more precise estimates over the global error of the numerical method.

Let consider an explicit Runge-Kutta method [48], i.e. we don't have to solve any equation at each iteration of the method, and let consider its corresponding *Butcher tableau*:

0			
c_2	$a_{2,1}$		(0)
$\vdots$	$\vdots$	$\ddots$	
c_s	$a_{s,1}$	$\dots$	$a_{s,s-1}$
	b_1	$\dots$	b_{s-1}

In order to approximate the modified field, we use a neural network of the form (1.3) and follow the global strategy introduced in the previous subsection. Let consider the Runge-Kutta method with the previous Butcher tableau, applied to the field $f_\theta(\cdot, h)$.

Theorem 25. *Let consider the learning error δ and the global error e_n at time $t_n := nh$ defined in theorem 24. Let consider*

$$\lambda \;:=\; \underset{h \in H}{Max} \, ||df_\theta(\cdot, h)||_{L^\infty(\Omega)} \tag{1.11}$$

and denoting the lower triangular matrix $A = [a_{i,j-1}]_{2 \leqslant j \leqslant i \leqslant s} \in \mathcal{M}_{s-1}(\mathbb{R})$ and the column vector $b = (b_j)_{1 \leqslant j \leqslant s} \in \mathbb{R}^s$ endowed of following norms for A and b given by

$$||A||_\infty = \underset{2 \leqslant i \leqslant s}{Max} \sum_{j=1}^{s} |a_{i,j}| \,, \qquad ||b||_1 = \sum_{i=1}^{s} |b_i| \,. \tag{1.12}$$

Finally, assume the explicit Runge-Kutta method has an order p. We have

$$\underset{0 \leqslant n \leqslant N}{Max} |e_n^*| \;\leqslant\; \frac{\delta h^p}{\lambda} \left(e^{\lambda \alpha T} - 1 \right) \tag{1.13}$$

where:

$$\alpha \;=\; ||b||_1 \left(1 + \lambda h_+ \, ||A||_\infty \, e^{\lambda h_+ ||A||_\infty}\right)$$

1.2.3 An alternative method for parallel training

In this subsection, we show how to learn the modified field in an alternative way. The main idea consists in training separately each term (say for instance of the modified field for the forward Euler method), by creating different data sets for different time steps $h_1 < \cdots < h_{N_h}$:

$$y_j \;=\; y_0 + h_j f(y_0) + h_j^2 f^{[1]}(y_0) + \cdots + h_j^{N_t} f^{[N_t-1]}(y_0) + h_j^{N_t+1} R(y_0, h_j). \quad (1.14)$$

Note that in contrast with previous method, the step-size is not chosen at random for each initial value. We then obtain a linear system which can be solved by using the *generalized inverse* of a matrix. The solution of this linear system encompasses the values of $f^{[1]}(y_0)$, ..., $f^{[N_t-1]}(y_0)$ and $R(y_0, h_1)$, ..., $R(y_0, h_{N_h})$ which correspond to data usable for learning each term of the modified field separately.

The main advantages of this method are a shorter training-time (at least on a parallel machine), thus allowing for a larger number of data, and a smaller computational time (again on a parallel machine) when it comes to obtaining the numerical solution from trained values.

1.3 Numerical experiments

In order to illustrate our theoretical results, we have tested the method given in Section 1.2.1 for two simple dynamical systems used from simple physics:

1. **The Non-linear Pendulum:** This system describes the movement of a pendulum under the influence of gravity. It is governed by the equations

$$\begin{cases} \dot{y_1} &= -\sin(y_2) \\ \dot{y_2} &= \quad y_1 \end{cases},$$

where y_2 denotes the angle of the pendulum with respect to the vertical and y_1 its angular velocity. Note that the system is Hamiltonian, see [32, 50, 59]. Parameters are given in Appendix 1.C.2.1 for the forward Euler method, Appendix 1.C.2.3 for the Runge-Kutta 2 method and Appendix 1.C.2.4 for the midpoint rule.

2. **The Rigid Body system:** This is a three-dimensional system which describes the angular rotation of a solid in the physical space

$$\left\{ \begin{array}{rcl} \dot{y_1} & = & \left(\frac{1}{I_3} - \frac{1}{I_2}\right) y_2 y_3 \\ \dot{y_2} & = & \left(\frac{1}{I_1} - \frac{1}{I_3}\right) y_1 y_3 \\ \dot{y_3} & = & \left(\frac{1}{I_2} - \frac{1}{I_1}\right) y_1 y_2 \end{array} \right. ,$$

where y_1, y_2 and y_3 denote the angular momenta, and I_1, I_2 and I_3 the momenta of inertia [50] (we take here $I_1 = 1$, $I_2 = 2$, $I_3 = 3$). It possesses two invariants, the so-called Casimir $C(y) = \frac{1}{2}|y|^2$ and the energy $H(y) = \frac{1}{2}\left(\frac{y_1^2}{I_1} + \frac{y_2^2}{I_2} + \frac{y_3^2}{I_3}\right)$. Hence, the solution lies at the intersection of the sphere $|y|^2 = |y(0)|^2$ and of the ellipsoïd $\frac{y_1^2}{I_1} + \frac{y_2^2}{I_2} + \frac{y_3^2}{I_3} = 2H(y(0))$. The domain Ω used for training is thus chosen accordingly. Parameters are given in Appendix 1.C.2.2.

For the *Forward Euler* and *RK2* methods, the modified field (1.2) can be computed by recursive formulae based on various derivatives of f, see for instance [20, 50]. It is represented by a series whose general term $f^{[j]}$ has an explicit -though complicated- expression, which can be compared with its numerical counterpart, obtained by learning it from the data set. For all $y \in \mathbb{R}^d$, $1 \leqslant j \leqslant k-1$, we have on the one hand

$$f^{[1]}(y) = \frac{1}{2} df(y) f(y) \tag{1.15}$$

$$f^{[j]}(y) = \frac{1}{j+1} df^{[j-1]}(y) f(y) \tag{1.16}$$

for the Forward Euler method, and on the other hand

$$f^{[1]}(y) = \frac{1}{24} d(df \cdot f)(y) f(y) + \frac{1}{8} df(y)^2 f(y) \tag{1.17}$$

$$f^{[2]}(y) = \frac{1}{24} d\left(d\left(df \cdot f\right) \cdot f\right) f(y) - \frac{1}{2} df(y) f^{[1]}(y) - \frac{1}{2} df^{[1]}(y) f(y). \tag{1.18}$$

for the Runge-Kutta 2 method. Formulas associated to the midpoint method are given in [20].

Truncating the formal power series (1.2) then gives an approximation of the theoretical modified field, which serves as a reference

$$\tilde{f}_h(y) = f(y) + h^p \sum_{j=1}^{k-1} h^{j-1} f^{[j]}(y) + \mathcal{O}\left(h^{k+p-1}\right). \tag{1.19}$$

Here, p is the order of the numerical method under consideration.

1.3.1 Approximation of the modified field

In this subsection, we study the approximation error between the learned modified field (1.3) and the theoretical modified field (1.2) for the nonlinear Pendulum. We observe the learning error w.r.t. both space and time step variables. More precisely, we plot the function

$$g_h^k : x \;\mapsto\; \frac{1}{h^p} \left| \tilde{f}_h^{\,k}(x) - f_\theta(x,h) \right| \tag{1.20}$$

for $k = 4$ over the domain $\Omega = [-2,2]^2$ in order to study the learning error in space, where the $\mathcal{O}$-term in $\tilde{f}_h(y)$ is simply neglected. We furthermore represent $\underset{\Omega}{\mathrm{Max}} g_h^k$ for several values of time steps h, in order to study the learning error in function of the the time step. Note that we clearly get the expected order of convergence of $f_\theta(\cdot, h)$ towards the modified field $\tilde{f}_h$, with the exception of a plateau for small values of h .

Figures 1.1,1.2 and 1.3 show that the error g_h^4 is globally constant at the center of the domain and grows near its boundaries (see [32] where a similar behaviour is observed).

Altogether, these experiments confirm that the modified field can be appropriately learned with our neural network.

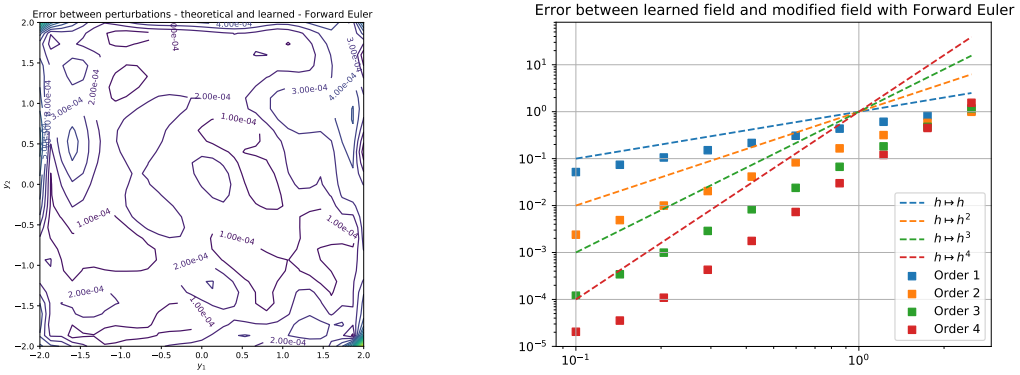

Figure 1.1 – Forward Euler method. Left: Difference between $\tilde{f}_h^{\,4}$ and $f_\theta(\cdot, h)$ for $h = 0.1$. Right: Error between $\tilde{f}_h^{\,k}$ and $f_\theta(\cdot, h)$ for $1 \leqslant k \leqslant 4$.

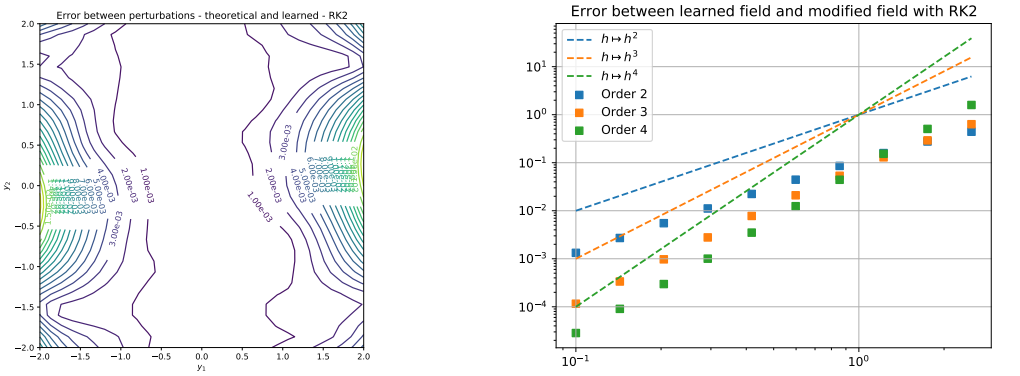

Figure 1.2 – Runge-Kutta 2 method. Left: Difference between $\tilde{f}_h^{\,4}$ and $f_\theta(\cdot, h)$ for $h = 0.1$. Right: Error between $\tilde{f}_h^{\,k}$ and $f_\theta(\cdot, h)$ for $1 \leqslant k \leqslant 4$.

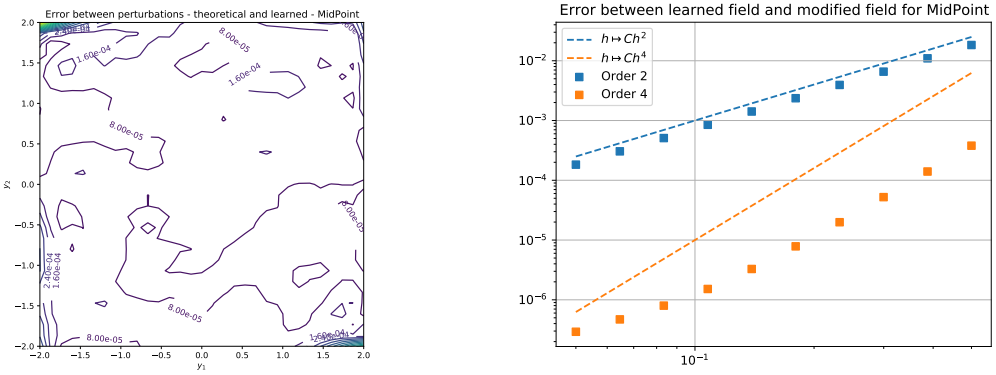

Figure 1.3 – Midpoint method. Left: Difference between $\tilde{f}_h{}^4$ and $f_\theta(\cdot, h)$ for $h = 0.05$. Right: Error between $\tilde{f}_h{}^k$ and $f_\theta(\cdot, h)$ for $1 \leqslant k \leqslant 4$. Note that owing to the structure of the modified field for midpoint method (see [20, 50]), terms for odd powers of h vanish.

1.3.2 Loss decay and Integration of ODE's

Now, in order to compare, for a given method, the integration of a dynamical system with the original field and with the learned modified field, we will solve the nonlinear Pendulum with the Forward Euler, Runge-Kutta 2 and midpoint methods and the Rigid Body system with the Forward Euler method. However, prior to that, we study the decays of the *Loss*-functions for the training and testing data sets ($Loss_{Train}$ and $Loss_{Test}$). Their similarity is a good indication that there is no overfitting (the size of the training data set is thus appropriately estimated). As the MSE *Loss* is used, it gives an idea of the value of the square of the learning error.

Figures 1.4, 1.5 and 1.6 show a more accurate numerical integration by using the corresponding learned modified field $f_\theta(\cdot, h)$ than using f. Moreover, exact flow and numerical flow with $f_\theta(\cdot, h)$ seem identical due to the small numerical error.

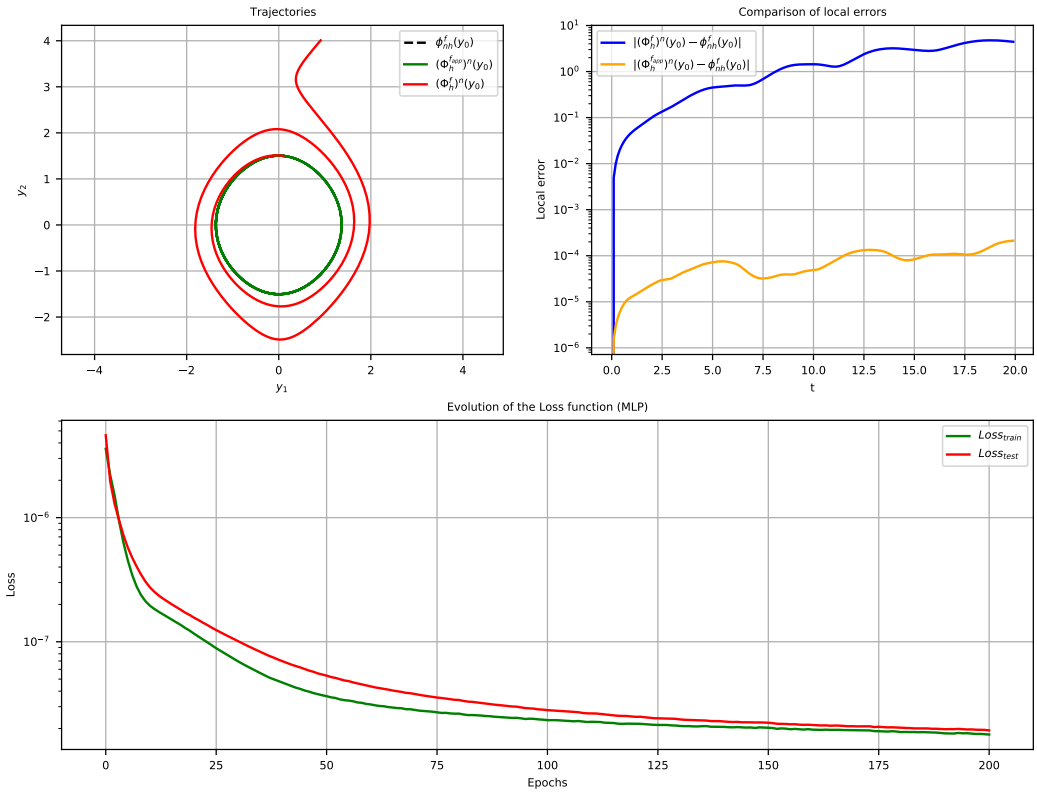

Figure 1.4 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, red: numerical flow with f, green: numerical flow with $f_\theta(\cdot, h)$) and local error (blue: exact flow and numerical flow with f, yellow: exact and numerical flow with $f_\theta(\cdot, h)$) for the nonlinear pendulum with Forward Euler method.

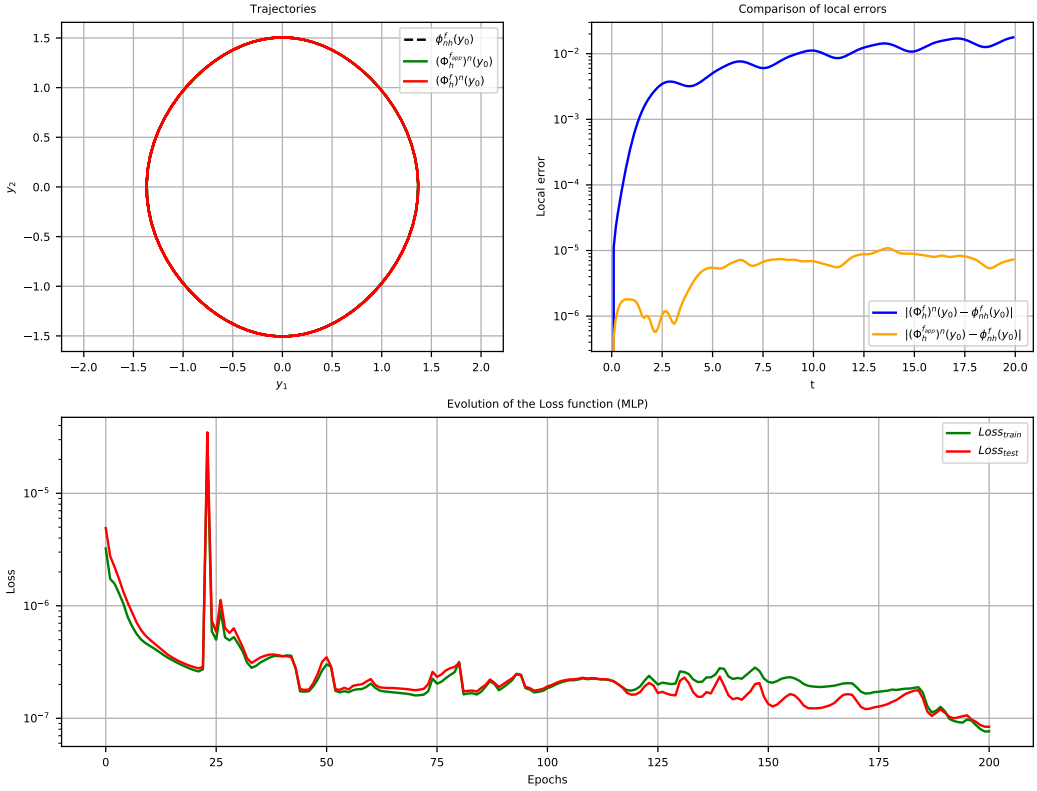

Figure 1.5 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, red: numerical flow with f, green: numerical flow with $f_\theta(\cdot, h)$) and local error (blue: exact flow and numerical flow with f, yellow: exact and numerical flow with $f_\theta(\cdot, h)$) for the nonlinear pendulum with Runge-Kutta 2 method.

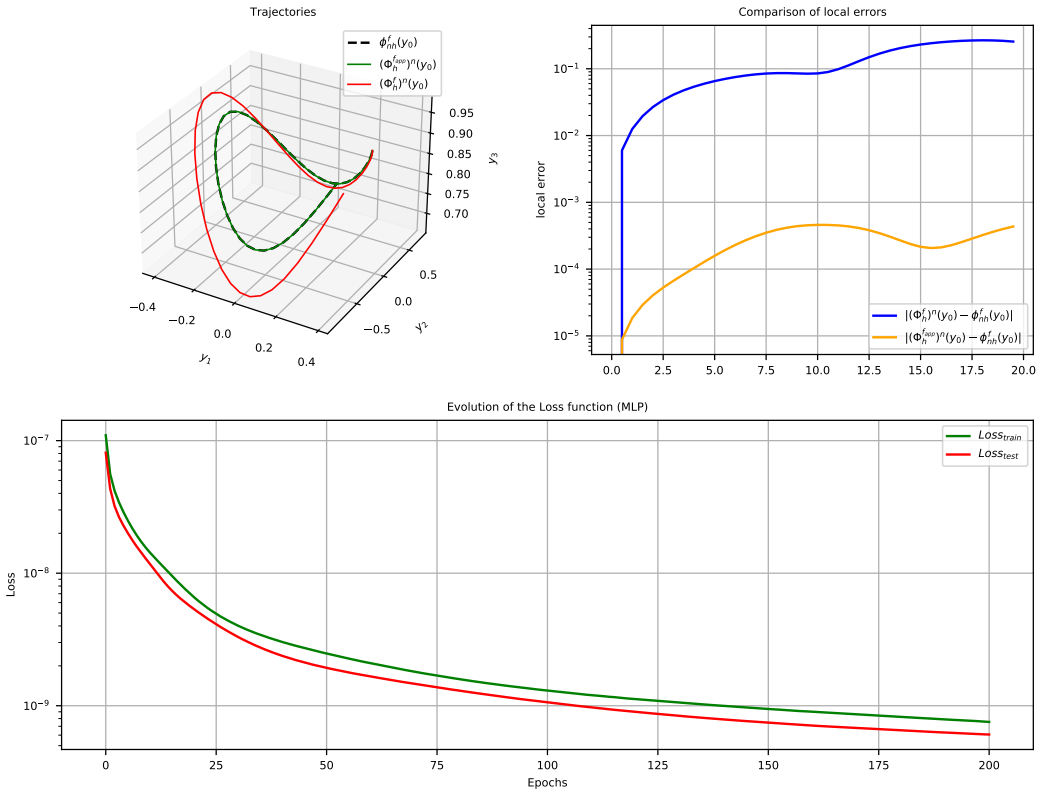

Figure 1.6 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, red: numerical flow with f, green: numerical flow with $f_\theta(\cdot, h)$) and local error (blue: exact flow and numerical flow with f, yellow: exact and numerical flow with $f_\theta(\cdot, h)$) for the Rigid Body system with Forward Euler method.

As the midpoint method is a symmetric and symplectic method, it is known to preserve accurately the geometric properties of the model. In order to evaluate the extent to which this feature persists in our context, we simply plot the value of the Hamiltonian along the numerical solution obtained from learned data. We test this method for the pendulum system, which is hamiltonian. Figure 1.7 shows a smaller error for integration with $f_\theta(\cdot, h)$ by using the midpoint method than integration with f. Moreover, the hamiltonian function of the pendulum system, given by

$$H : y \quad \mapsto \quad \frac{1}{2}y_1^2 + (1 - \cos(y_2)) \tag{1.21}$$

is preserved by the midpoint method with $f_\theta(\cdot, h)$ with smaller oscillations than midpoint with f. Preservation is better than DOPRI5 too, which is a non-symplectic method, as shown in Figure 1.8.

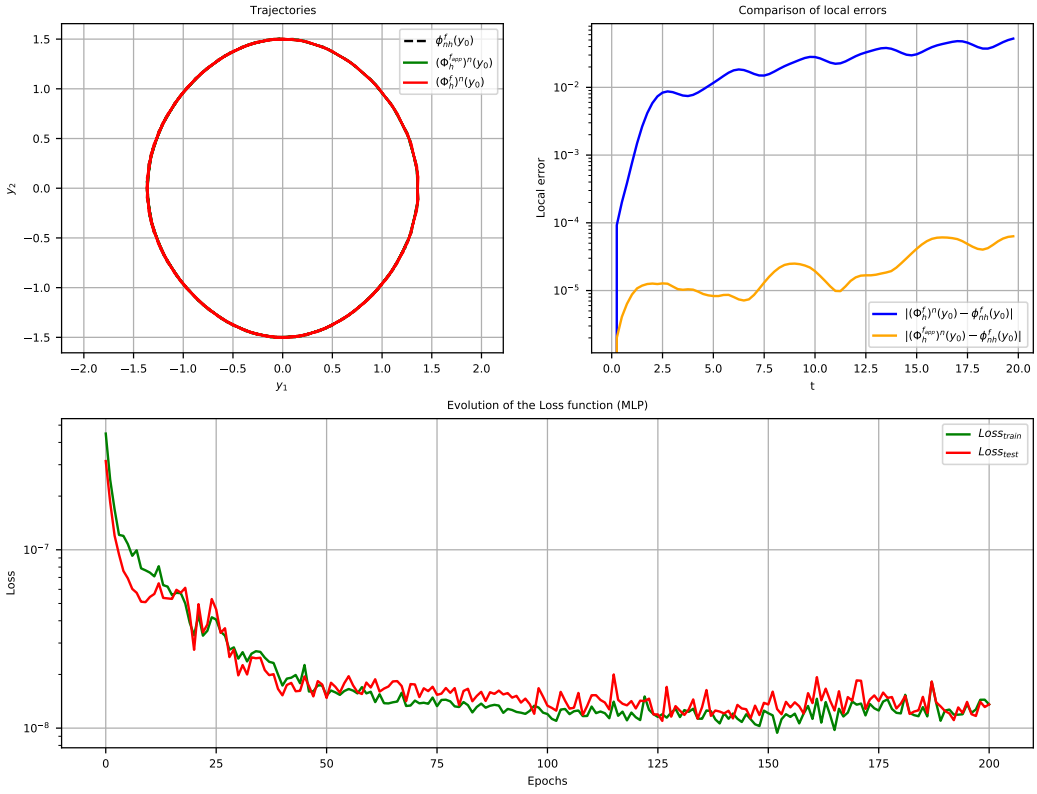

Figure 1.7 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, red: numerical flow with f, green: numerical flow with $f_\theta(\cdot, h)$) and local error (blue: exact flow and numerical flow with f, yellow: exact and numerical flow with $f_\theta(\cdot, h)$) for the nonlinear pendulum with midpoint method.

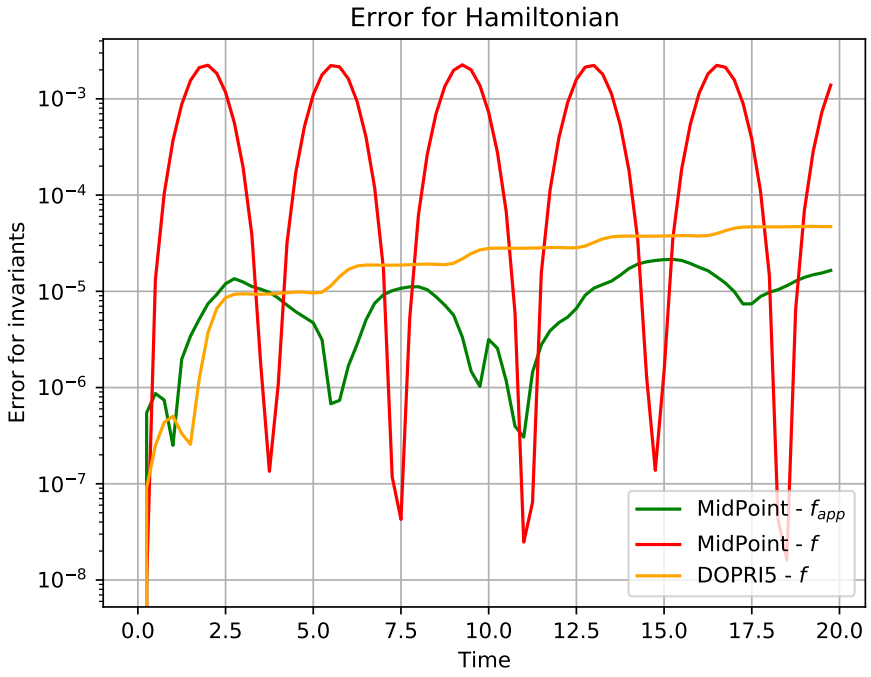

Figure 1.8 – Evolution of the error between Hamiltonian $H : y \mapsto (1-\cos(y_2)) + \frac{1}{2}y_1^2$ over the numerical flow and Hamiltonian at $t = 0$, $H(y_0)$.

Eventually, we study the global error between the exact flow and the approximation obtained from the original field, as well as the error between the exact flow and the numerical flow obtained form the learned modified field. The errors are plot as functions of the step-size and the curves are in perfect agreement with the estimates of previous theorems (for the Forward Euler, Runge-Kutta 2 and midpoint methods).

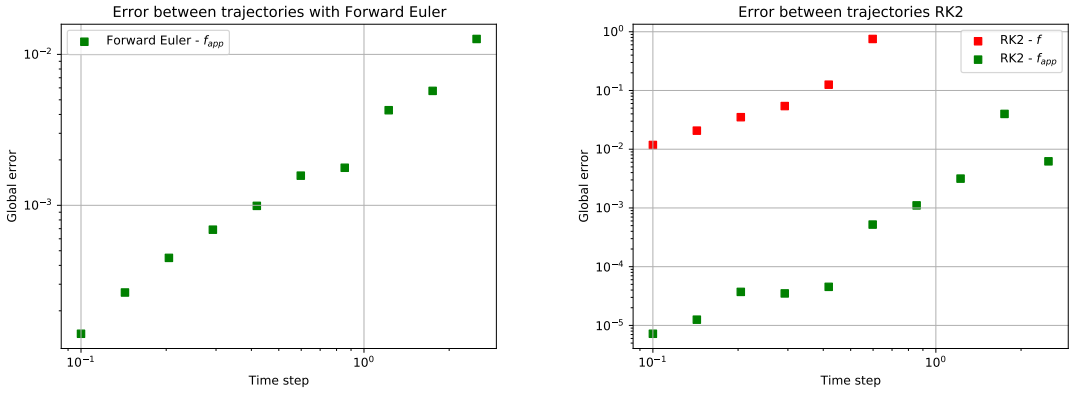

Figure 1.9 – Integration errors (green: integration with f, red: integration with $f_\theta(\cdot, h)$). Left: Nonlinear Pendulum with Forward Euler. Right: Nonlinear Pendulum with Runge-Kutta 2.

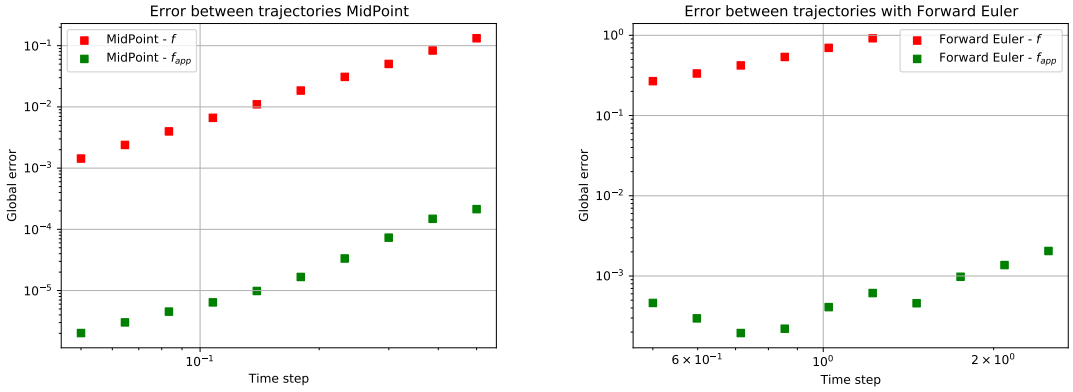

Figure 1.10 – Integration errors (green: integration with f, red: integration with $f_\theta(\cdot, h)$). Left: midpoint method for the nonlinear Pendulum. Right: Forward Euler method for the Rigid Body System.

Note that the estimate of Theorem 24 is confirmed by Figures 1.9 and 1.10 , with a smaller multiplicative constant for the RK2 method.

1.3.3 Computational times for explicit methods

In this subsection, we plot efficiency curves (global error w.r.t. computational time). As the main goal of this paper is to design cheaper and/or more accurate

solvers, we shall compare our results with the state-of-the-art Dormand & Prince methods [48].

Figures 1.11 and 1.12 show numerical errors for explicit methods (Forward Euler and Runge-Kutta 2) with $f_\theta(\cdot, h)$ can be smaller than numerical errors for DOPRI5 with f, especially for large time steps.

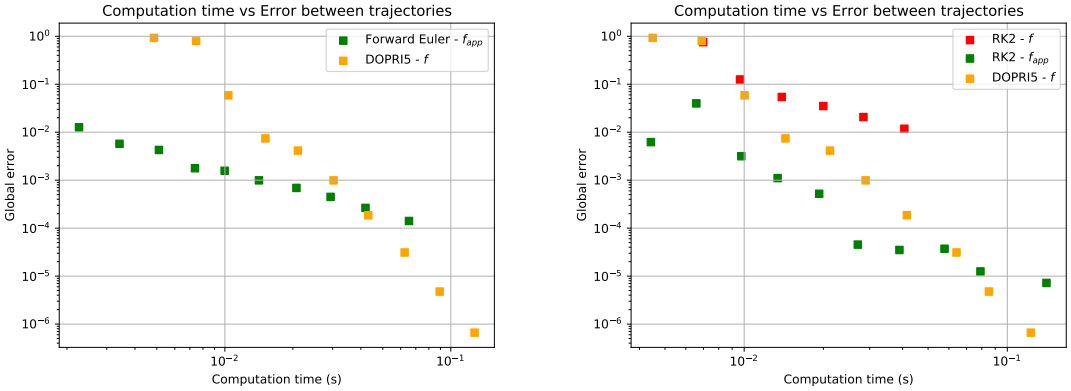

Figure 1.11 – Comparison between computational time and integration error (red: numerical method with f, green: integration with $f_\theta(\cdot, h)$, yellow: integration with DOPRI5). Left: Nonlinear Pendulum with Forward Euler. Right: Nonlinear Pendulum with Runge-Kutta 2.

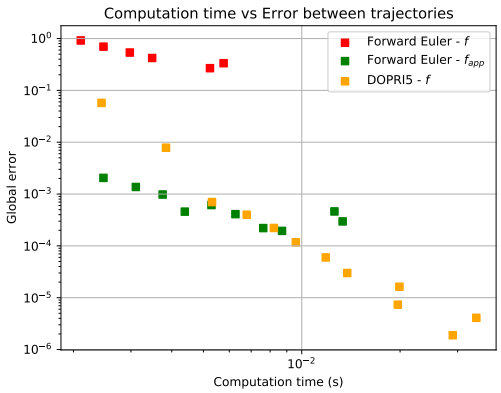

Figure 1.12 – Comparison between computational time and integration error (red: numerical method with f, green: integration with $f_\theta(\cdot, h)$, yellow: integration with DOPRI5) for Rigid Body system with Forward Euler method.

Besides, we compare the integration error with the learned modified field and the integration error using truncated modified field at the order k $\tilde{f}_h{}^k$. For the forward Euler method, use this numerical method with $\tilde{f}_h{}^k$ will give a numerical method of order k, called *modified Euler* [32, 50].

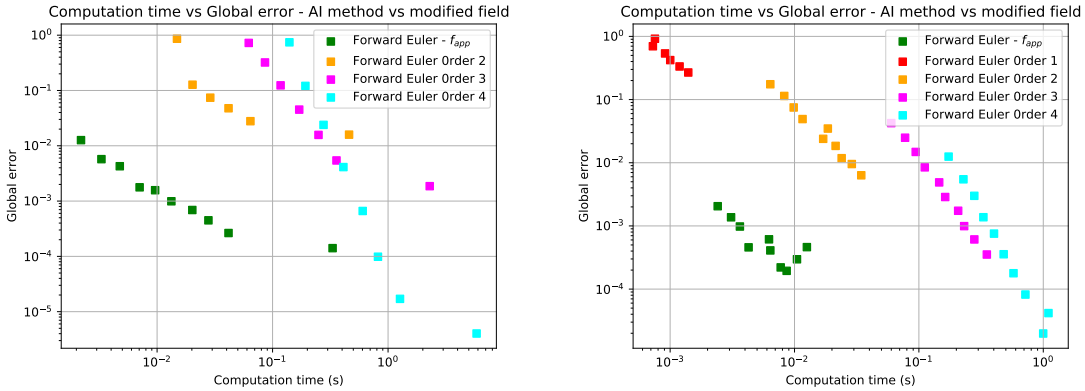

Figure 1.13 – Computational time versus integration error for the Forward Euler method (red: with f, yellow: with $\tilde{f}_h{}^2$, magenta: with $\tilde{f}_h{}^3$, cyan: with $\tilde{f}_h{}^4$, green: with $f_\theta(\cdot, h)$. Left: Nonlinear Pendulum. Right: Rigid Body system.

Figure 1.13 shows for both systems that integration with $\tilde{f}_h{}^k$ is more expensive in time or less accurate that integration with $f_\theta(\cdot, h)$. Moreover, we always conserve an interesting integration error whereas the other methods are not accurate at all for small computational time.

1.3.4 Evaluation of an alternative method

In this subsection, a comparison between the two learning techniques is illustrated. For the comparison to be fair, we adapt the volume of data so that the training times are nearly identical.

Figure 1.14 shows that both methods are able to generalize for smaller time steps than those used for training. In the two cases, traditional and alternative method with parallel training give approximately identical results

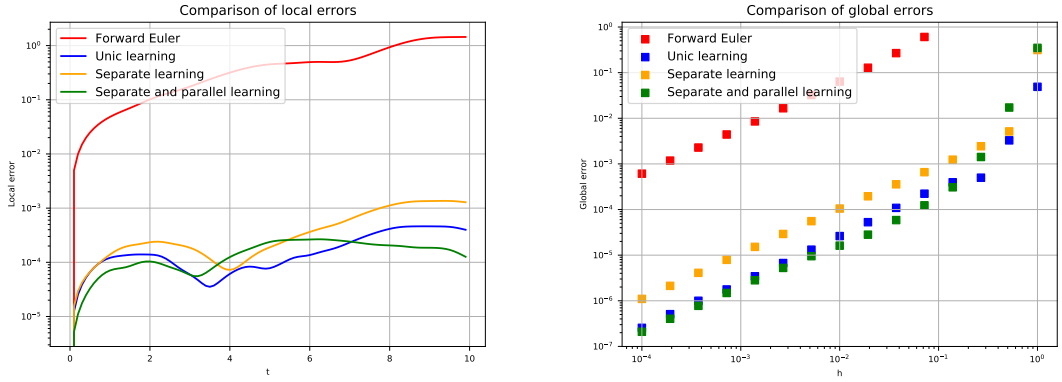

Figure 1.14 – Comparison between forward Euler, traditional and alternative method for simple pendulum. Left: Comparison of local errors. Right: Comparison of integration errors.

1.4 Conclusions

The numerical experiments presented in this paper demonstrate that learning the modified vector field is very beneficial in terms of efficiency. Clearly, the training of the network is an overload which should be considered separately for real-time applications. Another interesting outcome of our study is the fact that even when explicit formulae of the various terms of the modified field are available, it is advantageous to use its learned counterpart. Eventually, even though the technique used here is not fitted to situations where invariants should be conserved (learning directly the modified Hamiltonian would definitely be a better option), the learned vector field approximately retains the properties of its exact counterpart. It remains to be emphasized that the problems that we solved here are of small dimension and further studies with larges systems (for instance originating from PDEs) are needed.

1.A Appendix: Proof of Theorem 24

Let us consider, for time step h and for all $n \in [\![0, N]\!]$ (where $h = T/N$), the numerical scheme

$$y_0^* = y_0, \qquad y_{n+1}^* = \Phi_h^{f_\theta(\cdot,h)}\left(y_n^*\right) \tag{1.22}$$

The consistency error is of the form

$$\begin{aligned} \varepsilon_n^* &:= y(t_{n+1}) - \Phi_h^{f_\theta(\cdot,h)}\left(y(t_n)\right) \\ &= \underbrace{y(t_{n+1}) - \Phi_h^{\tilde{f}_h}\left(y(t_n)\right)}_{=0 \text{ (Modified field)}} + \Phi_h^{\tilde{f}_h}\left(y(t_n)\right) - \Phi_h^{f_\theta(\cdot,h)}\left(y(t_n)\right) \\ &= \Phi_h^{\tilde{f}_h}\left(y(t_n)\right) - \Phi_h^{f_\theta(\cdot,h)}\left(y(t_n)\right) \end{aligned} \tag{1.23}$$

so that, taking (1.7) and (1.8) into account, we have

$$|\varepsilon_n^*| \leqslant C\delta h^{p+1}.$$

Upon using (1.6), (1.22) and (1.23), the local truncation error can then be written as

$$e_{n+1}^* = \Phi_h^{f_\theta(\cdot,h)}(y_n^*) - \Phi_h^{f_\theta(\cdot,h)}(y(t_n)) - \varepsilon_n^*,$$

and from (1.9), we get

$$\left|e_{n+1}^*\right| \leqslant \left(1 + \overline{L}h\right)|e_n^*| + |\varepsilon_n^*| \leqslant \left(1 + \overline{L}h\right)|e_n^*| + C\delta h^{p+1}$$

where $\overline{L} := \underset{h\in[h_-,h_+]}{Max} L_{f_\theta(\cdot,h)}$. A discrete Grönwall lemma then leads to

$$|e_n^*| \leqslant C\delta h^{p+1}\sum_{j=0}^{n-1} e^{\overline{L}(n-j-1)h} \leqslant C\delta h^{p+1}\frac{e^{\overline{L}nh}-1}{e^{\overline{L}h}-1} \leqslant \frac{C\delta h^p}{\overline{L}}\left(e^{\overline{L}T}-1\right).$$

1.B Appendix: Proof of Theorem 25

Let introduce some notations. Let $n \in [\![0, N]\!]$:

$$
\begin{aligned}
\tilde{k_1} &= \tilde{f}_h(y(t_n)) \\
\tilde{k_2} &= \tilde{f}_h(y(t_n) + ha_{2,1}\tilde{k_1}) \\
&\vdots \\
\tilde{k_s} &= \tilde{f}_h\left(y(t_n) + h(a_{s,1}\tilde{k_1} + \ldots + a_{s,s-1}\tilde{k_{s-1}})\right) \\
\overline{k_1} &= f_\theta(y(t_n), h) \\
\overline{k_2} &= f_\theta(y(t_n) + ha_{2,1}\overline{k_1}, h) \\
&\vdots \\
\overline{k_s} &= f_\theta\left(y(t_n) + h(a_{s,1}\overline{k_1} + \ldots + a_{s,s-1}\overline{k_{s-1}}), h\right) \\
k_1^* &= f_\theta(y_n^*, h) \qquad (1.24) \\
k_2^* &= f_\theta(y_n^* + ha_{2,1}k_1^*, h) \\
&\vdots \\
k_s^* &= f_\theta\left(y_n^* + h(a_{s,1}k_1^* + \ldots + a_{s,s-1}k_{s-1}^*), h\right) \qquad (1.25)
\end{aligned}
$$

(i). Consistency error:

$$
\begin{aligned}
\varepsilon_n^* &:= y(t_{n+1}) - y(t_n) - h\sum_{i=1}^{s} b_i\overline{k_i} \qquad (1.26) \\
\varepsilon_n^* &= \underbrace{y(t_{n+1}) - y(t_n) - h\sum_{i=1}^{s} b_i\tilde{k_i}}_{=0 \text{ (Modified field)}} + h\sum_{i=1}^{s} b_i\tilde{k_i} - h\sum_{i=1}^{s} b_i\overline{k_i} \\
&= h\sum_{i=1}^{s} b_i\left(\tilde{k_i} - \overline{k_i}\right)
\end{aligned}
$$

By (1.7), $\left|\tilde{k_1} - \overline{k_1}\right| \leqslant \delta h^p$, then, for all $i \geqslant 2$:

$$\begin{aligned}
\tilde{k_i} - \overline{k_i} &= \tilde{f_h}\left(y(t_n) + h(a_{i,1}\tilde{k_1} + \ldots + a_{i,i-1}\tilde{k_{i-1}})\right) \\
&- f_\theta\left(y(t_n) + h(a_{i,1}\tilde{k_1} + \ldots + a_{i,i-1}\tilde{k_{i-1}}), h\right) \\
&+ f_\theta\left(y(t_n) + h(a_{i,1}\tilde{k_1} + \ldots + a_{i,i-1}\tilde{k_{i-1}}), h\right) \\
&- f_\theta\left(y(t_n) + h(a_{i,1}\overline{k_1} + \ldots + a_{i,i-1}\overline{k_{i-1}}), h\right)
\end{aligned}$$

With (1.7) and (1.11) (applied respectively for the first and second part of the expansion), we have:

$$\left|\tilde{k_i} - \overline{k_i}\right| \leqslant \delta h^p + \lambda h \sum_{j=1}^{i-1} |a_{i,j}| \left|\tilde{k_j} - \overline{k_j}\right|$$

Discrete Grönwall lemma leads to:

$$\begin{aligned}
\left|\tilde{k_i} - \overline{k_i}\right| &\leqslant \delta h^p + \sum_{j=1}^{i-1} \delta\lambda h^{p+1} |a_{i,j}| \, e^{\sum_{l=j+1}^{i-1} \lambda h |a_{i,l}|} \\
&\leqslant \delta h^p \left(1 + \lambda h \, ||A||_\infty \, e^{\lambda h ||A||_\infty}\right)
\end{aligned}$$

Therefore:

$$|\varepsilon_n^*| \leqslant \delta\alpha h^{p+1}$$

(ii). Local truncation error:

With (1.24), (1.6) and (1.26), we have:

$$e_{n+1}^* = e_n^* + h \sum_{i=1}^{s} b_i \left(\overline{k_i} - k_i^*\right) - \varepsilon_n^*$$

and:

$$\left|e_{n+1}^*\right| \leqslant |e_n^*| + h\sum_{i=1}^{s} |b_i| \left|\overline{k_i} - k_i^*\right| + \delta\alpha h^{p+1}$$

By (1.11), $\left|\overline{k_1} - k_1^*\right| \leqslant \lambda|e_n^*|$. Then, for all $i \geqslant 2$:

$$\begin{aligned}\overline{k_i} - k_i^* &= f_\theta\left(y(t_n) + h(a_{i,1}\overline{k_1} + \ldots + a_{i,i-1}\overline{k_{i-1}}), h\right) \\ &- f_\theta\left(y_n^* + h(a_{i,1}k_1^* + \ldots + a_{i,i-1}k_{i-1}^*), h\right)\end{aligned}$$

By (1.11), we have:

$$\left|\overline{k_i} - k_i^*\right| \leqslant \lambda|e_n^*| + \lambda h\sum_{j=1}^{i-1} |a_{i,j}| \left|\overline{k_j} - k_j^*\right|$$

Discrete Grönwall lemma leads to:

$$\begin{aligned}\left|\overline{k_i} - k_i^*\right| &\leqslant \lambda\left|e_n^*\right| + \sum_{j=1}^{i-1} \lambda^2 h \left|e_n^*\right| |a_{i,j}| \, e^{\sum_{l=j+1}^{i-1} \lambda h |a_{i,l}|} \\ &\leqslant \lambda\left|e_n^*\right| \left(1 + \lambda h \left|\left|A\right|\right|_\infty e^{\lambda h ||A||_\infty}\right)\end{aligned}$$

Therefore:

$$\left|e_{n+1}^*\right| \leqslant (1 + \lambda h\alpha)\left|e_n^*\right| + \delta\alpha h^{p+1}$$

By applying discrete Grönwall lemma, we have:

$$\begin{aligned} |e_n^*| &\leqslant \delta\alpha h^{p+1} \sum_{j=0}^{n-1} e^{\lambda\alpha(n-j-1)h} \\ &\leqslant \delta\alpha h^{p+1} \frac{e^{\lambda\alpha nh} - 1}{e^{\lambda\alpha h} - 1} \\ &\leqslant \frac{\delta h^p}{\lambda} \left(e^{\lambda\alpha T} - 1\right) \end{aligned}$$

1.C Appendix: Choice of the parameters

1.C.1 Link between learning error and parameters

The influence of the number of parameters over the learning error has been studied. In particular, we have studied the effect of the number of neurons and hidden layers. The dynamical system chosen for the test is the non-linear pendulum while the numerical method is simply the Forward Euler method. In order to approximate the learning error, we compute the value

$$\delta \approx \underset{h\in H^*}{Max} \frac{1}{h} \left|\left| \tilde{f}_h^{\,4}(x_{i,j}) - f_\theta(x_{i,j}, h) \right|\right|_{l^\infty} \tag{1.27}$$

where $(x_{i,j})_{0\leqslant i,j\leqslant 40}$ is a uniform grid on the square $\Omega = [-2,2]^2$, $H^* = (e^{h_j^*})_{0\leqslant j\leqslant 14}$ where $(h_j^*)_{0\leqslant j\leqslant 14}$ is a uniform discretization of $[\log(h^-), \log(h^+)]$ and $\tilde{f}_h^{\,4}$ corresponds to the field (1.2) for $k = 4$ with R neglected, computed via the formulas given in [20, 50].

As observed in Figure 1.15, a plateau appears when the number of parameters is large. This plateau has a lower value for a larger number of data. Moreover, we observe that deep networks are more efficient than shallow networks in order to learn the good vector field.

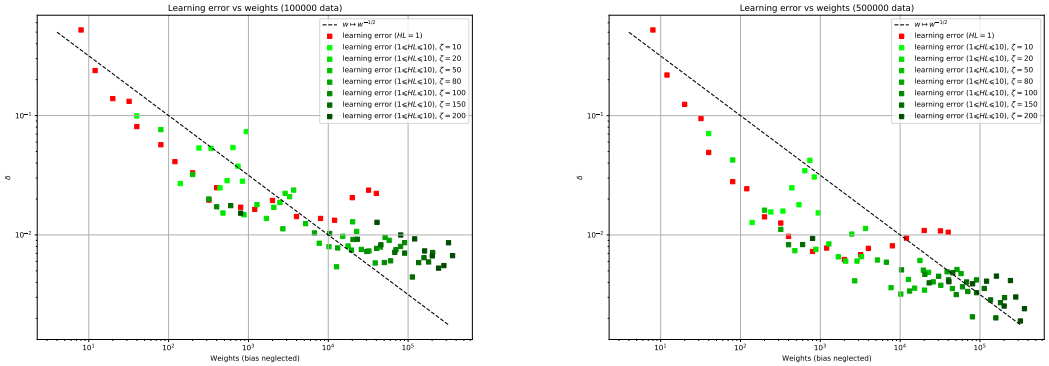

Figure 1.15 – Learning error δ versus number of parameters w in the neural network (number of neurons ζ and hidden layers HL). Shallow network is plot with red points whereas deep network is plot with green points (light green for 10 neurons to dark green for 200 neurons). Bias are neglected. Learning error is plot for 100 000 data (left) and 500 000 data (right). 200 epochs are used. $[h^-, h^+] = [10^{-2}, 10^{-1}]$. The curve of $w \mapsto w^{\frac{1}{2}}$ has been added for comparison purposes.

1.C.2 Parameters selected in our numerical experiments

For the training, the optimizer *Adam* of Pytorch is used. Besides, the mini-batching option is activated as it appears to be more efficient. Hyperbolic tangent tanh was selected as activation function.

1.C.2.1 Nonlinear Pendulum - Forward Euler

Parameters	
# Math Parameters:	
Dynamical system:	Pendulum
Numerical method:	Forward Euler
Interval where time steps are selected:	$[h_-, h_+] = [0.1, 2.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.1$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 25\,000\,000$
Proportion of data for training:	80% - $K_0 = 20\,000\,000$
Number of terms in the perturbation (MLP's):	$N_t = 1$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	300
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training: 10 h 14 min 17 s

1.C.2.2 Rigid body system - Forward Euler

Casimir invariant introduced at the beginning of the section 1.3 allows to chose the training data in the spherical crown $\{x \in [-2,2]^2 : 0.98 \leqslant |x| \leqslant 1.02\}$

Parameters	
# Math Parameters:	
Dynamical system:	Rigid Body
Numerical method:	Forward Euler
Interval where time steps are selected:	$[h_-, h_+] = [0.5, 2.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.5$
# AI Parameters:	
Domain where data are selected:	$\Omega = \left\{x \in [-2,2]^2 : 0.98 \leqslant \lvert x\rvert \leqslant 1.02\right\}$
Number of data:	$K = 100\,000\,000$
Proportion of data for training:	80% - $K_0 = 80\,000\,000$
Number of terms in the perturbation (MLP's):	$N_t = 1$
Hidden layers per MLP:	2
Neurons on each hidden layer:	250
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	300
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training: 1 Day 21 h 59 min 51 s

1.C.2.3 Nonlinear Pendulum - Runge-Kutta 2

Parameters	
# Math Parameters:	
Dynamical system:	Pendulum
Numerical method:	RK2
Interval where time steps are selected:	$[h_-, h_+] = [0.1, 2.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.1$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2,2]^2$
Number of data:	$K = 100\,000\,000$
Proportion of data for training:	80% - $K_0 = 80\,000\,000$
Number of terms in the perturbation (MLP's):	$N_t = 1$
Hidden layers per MLP:	2
Neurons on each hidden layer:	250
Learning rate:	$5 \cdot 10^{-4}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	300
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training: 3 Days 2 h 54 min 35 s

1.C.2.4 Nonlinear Pendulum - midpoint

Parameters	
# Math Parameters:	
Dynamical system:	Pendulum
Numerical method:	midpoint
Interval where time steps are selected:	$[h_-, h_+] = [0.05, 0.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.25$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 20\,000\,000$
Proportion of data for training:	80% - $K_0 = 16\,000\,000$
Number of terms in the perturbation (MLP's):	$N_t = 1$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	300
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training: 9 h 47 min 51 s

1.C.3 Nonlinear Pendulum - Comparison between traditional and an alternative method

For the standard method, data are created by simulating several solutions with the same initial condition for different time steps.

Parameters	
# Math Parameters:	
Dynamical system:	Pendulum
Numerical method:	Forward Euler
Interval where time steps are selected:	$[h_-, h_+] = [0.01, 0.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.1$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^2$
Number of data	
- Traditional method:	$(K, N_h) = (50\,000, 5)$
- Alternative method:	$(K, N_h) = (76\,129, 5)$
- Alternative method (parallel training):	$(K, N_h) = (105\,735, 5)$
Proportion of data for training:	80%
- Traditional method:	$(K_0, N_h) = (40\,000, 5)$
- Alternative method:	$(K_0, N_h) = (60\,903, 5)$
- Alternative method (parallel training):	$(K_0, N_h) = (84\,588, 5)$
Number of terms in the perturbation (MLP's):	$N_t = 3$
Hidden layers per MLP:	2
Neurons on each hidden layer:	50
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	100
Epochs:	200
Epochs between two prints of loss value:	20

For the alternative method with parallel training, the total computational time for training correspond to the maximum of all training times for each term of the modified field.

Computational time for training			
Method	Traditional	Alternative	Alternative (parallel training)
f_1		2 min 54 s	3 min 24 s
f_2		2 min 22 s	3 min 28 s
R		13 min 23 s	17 min 9 s
Total	**18 min 33 s**	**18 min 41 s**	**17 min 9 s**

Chapter 2

First order Integral scheme performing with Neural Network

In this chapter, we will now consider highly oscillatory differential equations. The main goal is to perform an existing numerical scheme in order to get a cheap and accurate method.

2.1 Introduction

Highly oscillatory differential equations are a particular case of multi-scale problems. For theoretical study, averaging theory can be used in order to study this kind of equations [24, 87]. Concerning numerical approximations, standard methods such as Runge-Kutta does not work, especially due to high oscillations which create an increasing of the convergence error. However, structure of equations and averaging theory have been used to develop uniformly accurate numerical methods [21, 22] whose error does not depend on oscillations by using a micro-macro decomposition or pullback decomposition, both based over averaging theory.

All these numerical methods require a lot of computations before numerical integration. Amongst the micro-macro decomposition exist integral schemes, which reduce computations before integration. However, only low order schemes does not require a lot of preliminary computations.

By using similar machine learning methods than those used for autonomous equations [10], we can perform a first order scheme in order to improve accuracy.

2.2 Improving the accuracy of a numerical scheme with machine learning

Consider an highly oscillatory differential equation of the form

$$\begin{cases} \dot{y^\varepsilon}(t) &= f\left(\frac{t}{\varepsilon}, y^\varepsilon(t)\right) \in \mathbb{R}^d, t \in [0,T], \varepsilon \in]0,1] \\ y^\varepsilon(0) &= y_0 \end{cases} \tag{2.1}$$

where $f : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$ is assumed to be smooth enough and 2π-periodic w.r.t. its first variable. By Cauchy-Lipshitz theorem, we have existence and uniqueness of a solution for any given initial value $y_0 \in \mathbb{R}^d$. We wish to approximate the solution over $[0,T]$ at times $t_n = nh, 0 \leqslant n \leqslant N$, where $h = \frac{T}{N}$ is the time step and N is the number of discretization points.

2.2.1 General strategy

As explained in the Introduction section, we shall approximate vector field with the help of neural network, which corresponds to the approximation of modified vector field in the autonomous case.

2.2.1.1 Forward Euler Integral scheme

Let consider an approximation $(y_n^\varepsilon)_{n\in\mathbb{N}}$ of the solution $y^\varepsilon(t_n)$, defined recursively by the formula

$$y_{n+1}^\varepsilon = y_n^\varepsilon + \int_{t_n}^{t_{n+1}} f\left(\frac{t}{\varepsilon}, y_n^\varepsilon\right) \mathrm{d}t \tag{2.2}$$

called *Forward Euler Integral scheme*. An error estimate shows that

$$\underset{0\leqslant n\leqslant N}{Max} |y_n^\varepsilon - y^\varepsilon(t_n)| \leqslant Ch \tag{2.3}$$

where constant C does not depend on ε. This scheme is uniformly accurate w.r.t. ε. Notice that this scheme does not require preliminary computations, but is only of order 1.

2.2.1.2 The help of backward error analysis

If we follow computations, this is possible to get the exact flow with this numerical scheme.

Proposition 26. *Let $p \in \mathbb{N}$ such that $p \geqslant 2$ and $t_0 \in \mathbb{R}$. We have T*

$$\begin{aligned} y^\varepsilon(t_0+h) &= y^\varepsilon(t_0) + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y^\varepsilon(t_0)\right) \mathrm{d}t \\ &+ \int_{t_0}^{t_0+h} \int_{t_0}^{t} \frac{\partial f}{\partial y}\left(\frac{t}{\varepsilon}, y^\varepsilon(t_0)\right) f\left(\frac{s}{\varepsilon}, y^\varepsilon(t_0)\right) \mathrm{d}s\mathrm{d}t \qquad (2.4) \\ &+ \sum_{k=2}^{p-1} \int_{t_0}^{t_0+h} \int_{t_0}^{\sigma_0} \cdots \int_{t_0}^{\sigma_{k-1}} g_k\left(\frac{\sigma_0}{\varepsilon}, \frac{\sigma_1}{\varepsilon}, \cdots, \frac{\sigma_k}{\varepsilon}, y^\varepsilon(t_0)\right) \mathrm{d}\sigma_k \cdots \mathrm{d}\sigma_1 \mathrm{d}\sigma_0 \\ &+ \int_{t_0}^{t_0+h} \int_{t_0}^{\sigma_0} \cdots \int_{t_0}^{\sigma_{p-1}} g_p\left(\frac{\sigma_0}{\varepsilon}, \frac{\sigma_1}{\varepsilon}, \cdots, \frac{\sigma_p}{\varepsilon}, y^\varepsilon(\sigma_p)\right) \mathrm{d}\sigma_p \cdots \mathrm{d}\sigma_1 \mathrm{d}\sigma_0 \end{aligned}$$

where $g_0(\tau_0, y) = f(\tau_0, y)$ and for all $k \geqslant 0$,

$$g_{k+1}(\tau_0, \cdots, \tau_k, \tau_{k+1}, y) = \frac{\partial g_k}{\partial y}(\tau_0, \cdots, \tau_k, y)\, f(\tau_{k+1}, y) \qquad (2.5)$$

Remarks. ***(i)*** *This formula corresponds to an extension of the modified equation in this non-autonomous case (highly oscillatory equation) with the Forward Euler scheme.*

(ii) *With this formula, if we replace t_0 by t_n, $y^\varepsilon(t_0+h)$ by y^ε_{n+1}, $y^\varepsilon(t_0)$ by y^ε_n and neglect the last remainder term, we get a scheme of order p. For example, in the case $p = 2$, let consider the scheme*

$$y^\varepsilon_{n+1} = y^\varepsilon_n + \int_{t_n}^{t_{n+1}} f\left(\frac{t}{\varepsilon}, y^\varepsilon_n\right) \mathrm{d}t + \int_{t_0}^{t_{n+1}} \int_{t_n}^{t} \frac{\partial f}{\partial y}\left(\frac{t}{\varepsilon}, y^\varepsilon_n\right) f\left(\frac{s}{\varepsilon}, y^\varepsilon_n\right) \mathrm{d}s\mathrm{d}t \qquad (2.6)$$

we have

$$\underset{0 \leqslant n \leqslant N}{Max} |y^\varepsilon_n - y^\varepsilon(t_n)| \leqslant Ch^2 \qquad (2.7)$$

where the constant C does not depend on ε.

2.2.1.3 Machine learning method

Let consider $\varepsilon \in]0, 1]$, fixed. As formula (2.4) leads to

$$\begin{aligned} y^\varepsilon(t_0+h) &= y^\varepsilon(t_0) \\ &+ \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y^\varepsilon(t_0)\right) \mathrm{d}t + h^2 R_p^\varepsilon\left(\cos\left(\frac{t_0}{\varepsilon}\right), \sin\left(\frac{t_0}{\varepsilon}\right), y^\varepsilon(t_0), h\right) \\ &+ \mathcal{O}\left(h^{p+1}\right). \end{aligned} \tag{2.8}$$

where $p \geqslant 2$ and R_p^ε corresponds to

$$\begin{aligned} & R_p^\varepsilon\left(\cos\left(\frac{t_0}{\varepsilon}\right), \sin\left(\frac{t_0}{\varepsilon}\right), y, h\right) \\ := & \frac{1}{h^2}\int_{t_0}^{t_0+h}\int_{t_0}^{t} \frac{\partial f}{\partial y}\left(\frac{t}{\varepsilon}, y^\varepsilon(t_0)\right) f\left(\frac{s}{\varepsilon}, y^\varepsilon(t_0)\right) \mathrm{d}s\mathrm{d}t \\ + & \frac{1}{h^2}\sum_{k=2}^{p-1}\int_{t_0}^{t_0+h}\int_{t_0}^{\sigma_0}\cdots\int_{t_0}^{\sigma_{k-1}} g_k\left(\frac{\sigma_0}{\varepsilon}, \frac{\sigma_1}{\varepsilon}, \cdots, \frac{\sigma_k}{\varepsilon}, y^\varepsilon(t_0)\right) \mathrm{d}\sigma_k\cdots\mathrm{d}\sigma_1\mathrm{d}\sigma_0. \end{aligned} \tag{2.9}$$

We can approximate the function R_p^ε by a neural network. More precisely, the function $(t, y, h) \mapsto R_p^\varepsilon\left(\cos\left(\frac{t}{\varepsilon}\right), \sin\left(\frac{t}{\varepsilon}\right), y, h\right)$ will be approximated by the function

$$(t, y, h) \mapsto R_\theta^\varepsilon\left(\cos\left(\frac{t}{\varepsilon}\right), \sin\left(\frac{t}{\varepsilon}\right), y, h\right) \tag{2.10}$$

where R_θ is a Multi Layer Perceptron (MLP). As in the autonomous case, the advantage of this structure is the consistency of the numerical method.

Remark. *As we have*

$$\begin{aligned} & \int_{t_0}^{t_0+h}\int_{t_0}^{\sigma_0}\cdots\int_{t_0}^{\sigma_{k-1}} e^{\frac{ij_0\sigma_0 + ij_1\sigma_1 + \cdots + ij_k\sigma_k}{\varepsilon}} \mathrm{d}\sigma_k\cdots\mathrm{d}\sigma_1\mathrm{d}\sigma_0 \\ \in & \ Vect\left\{\varepsilon^{\alpha_1} h^{\alpha_2} e^{\frac{i(\alpha_3 t_0 + \alpha_4 h)}{\varepsilon}}, (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \mathbb{N}^4\right\}, \end{aligned} \tag{2.11}$$

R_p^ε actually depends on $\cos\left(\frac{t_0}{\varepsilon}\right)$, $\sin\left(\frac{t_0}{\varepsilon}\right)$, $\cos\left(\frac{h}{\varepsilon}\right)$, $\sin\left(\frac{h}{\varepsilon}\right)$, h and ε. However, as ε is fixed, dependancy w.r.t. $\cos\left(\frac{h}{\varepsilon}\right)$ and $\sin\left(\frac{h}{\varepsilon}\right)$ is not mandatory for R_θ^ε.

The method consists of three main step. First, data are created by simulating the exact flow with an accurate (but expensive) numerical method. As in autonomous case, many data are required in order to get a good approximation of the remainder

function R^ε of the integral scheme. Then, the neural network is trained by minimizing the *Loss*-function. Finally, the integral scheme is used by replacing the unknown R^ε by the approximated function by neural network.

1. **Construction of the data set:** We take K initial data $y_0^{(k)}$, randomly selected into a compact set $\Omega \subset \mathbb{R}^d$ (assume that the flow remains into Ω over the time interval $[0, T]$), and $t_0^{(k)} \in [0, 2\pi]$, randomly selected too. Then, for all $k \in [\![0, K-1]\!]$, we compute a very accurate solution, which is a similar to the exact flow, from time $t_0^{(k)}$ to time $t_0^{(k)} + h^{(k)}$ with condition a time $t_0^{(k)}$ given by $y_0^{(k)}$. The solution at time $t_0^{(k)} + h^{(k)}$ is denoted $y_1^{(k)}$. Time step $h^{(k)}$ is randomly selected in the domain $[h_-, h_+]$ by using logarithm, as done in the autonomous case.

2. **Training of neural networks:** As in autonomous case, we minimise the MSE, which measures the error between predicted data $\hat{y_1}^{(k)}$ and "exact" data $y_1^{(k)}$ by computing the optimal neural network parameters over K_0 (where $K_0 \leqslant K$) data by using a gradient method:

$$Loss_{Train} = \frac{1}{K_0} \sum_{k=0}^{K_0-1} \frac{1}{h^{(k)4}} \left| y_1^{(k)} - \hat{y_1}^{(k)} \right|^2 \tag{2.12}$$

where y_1 is given by

$$y_1 = y_0 + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y_0\right) \mathrm{d}t + h^2 R_p^\varepsilon \left(\cos\left(\frac{t_0}{\varepsilon}\right), \sin\left(\frac{t_0}{\varepsilon}\right), y_0, h\right) + \mathcal{O}\left(h^{p+1}\right) \tag{2.13}$$

and $\hat{y_1}$ is given by

$$\hat{y_1} = y_0 + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y_0\right) \mathrm{d}t + h^2 R_\theta^\varepsilon \left(\cos\left(\frac{t_0}{\varepsilon}\right), \sin\left(\frac{t_0}{\varepsilon}\right), y_0, h\right) \tag{2.14}$$

At the same time, we copmpute the value of $Loss_{Test}$, wich measures the error between "exact" and predicted data which are not used for training:

$$Loss_{Test} = \frac{1}{K - K_0} \sum_{k=0}^{K_0-1} \frac{1}{h^{(k)4}} \left| y_1^{(k)} - \hat{y_1}^{(k)} \right|^2 \tag{2.15}$$

If $Loss_{Train}$ and $Loss_{Test}$ have the same decay order, the neural network is

well trained and able to generalize to other inputs as training data.

3. **Numerical approximation:** After the training, we get an accurate approximation of $y^\varepsilon(t_n)$ with the sequence $(y^\varepsilon_{\theta,n})_{n\in\mathbb{N}}$ defined recursively by

$$y^\varepsilon_{\theta,n+1} = y^\varepsilon_{\theta,n} + \int_{t_n}^{t_{n+1}} f\left(\frac{t}{\varepsilon}, y^\varepsilon_{\theta,n}\right) \mathrm{d}t + h^2 R^\varepsilon_\theta\left(\cos\left(\frac{t_n}{\varepsilon}\right), \sin\left(\frac{t_n}{\varepsilon}\right), y^\varepsilon_{\theta,n}, h\right) \quad (2.16)$$

2.2.2 Error analysis

In this subsection, we analyse the error resulting from the method described in the previous subsection. We give an error estimate associated to the improving of the Integral Forward Euler scheme.

Theorem 27. *Let consider the sequence $(y^\varepsilon_{\theta,n})_{n\in\mathbb{N}}$ defined with the formula (2.16) and let consider the global error*

$$e_{\theta,n} := y^\varepsilon_{\theta,n} - y^\varepsilon(t_n) \quad (2.17)$$

at times $t_n = nh$ for all $n \in [\![0, N]\!]$. Let $p \geqslant 2$. Let denote the learning error by

$$\delta_\varepsilon := \underset{(u,v,y,h)\in[-1,1]\times[-1,1]\times\Omega\times[0,h_+]}{Max} \left|R^\varepsilon_p(u,v,y,h) - R^\varepsilon_\theta(u,v,y,h)\right|. \quad (2.18)$$

Then there exist constants $\tilde{L}_\varepsilon, C > 0$ such that:

$$\underset{0\leqslant n\leqslant N}{Max} |e_{\theta,n}| \leqslant \frac{e^{\tilde{L}_\varepsilon T} - 1}{\tilde{L}_\varepsilon} \left[\delta_\varepsilon h + Ch^p\right] \quad (2.19)$$

Remark. *As ε is fixed before the beginning of the data creation and training. Therefore, we have to do a new training if we want to change the value of ε.*

2.3 Numerical experiment

In order to illustrate our theoretical result, method described in the subsection 2.2.1 is tested over a dynamical system.

The Inverted Pendulum: This system describes the movement a pendulum that has its center of mass above its pivot point. It is governed by the equations:

$$\left\{\begin{array}{rcl} \dot{y_1^\varepsilon}(t) & = & y_2^\varepsilon(t) + \sin\left(\frac{t}{\varepsilon}\right)\sin\left(y_1^\varepsilon(t)\right) \\ \dot{y_2^\varepsilon}(t) & = & \sin\left(y_1^\varepsilon(t)\right) - \frac{1}{2}\sin\left(\frac{t}{\varepsilon}\right)^2\sin\left(2y_1^\varepsilon(t)\right) - \sin\left(\frac{t}{\varepsilon}\right)\cos\left(y_1^\varepsilon(t)\right)y_2^\varepsilon(t) \end{array}\right. \tag{2.20}$$

2.3.1 Loss decay and Integration of ODE's

We compare the integration of the Inverted Pendulum system by using the Forward Euler integral scheme with the original field, and by adding the learned perturbation with neural network. We plus the MSE *Loss* decay (train and test). If both *Loss* functions have the same decay order, the neural network can be considered as well trained.

Figures 2.1, 2.2 and 2.3 show a more accurate integration by adding the learned perturbation with the neural network than using only the original field.

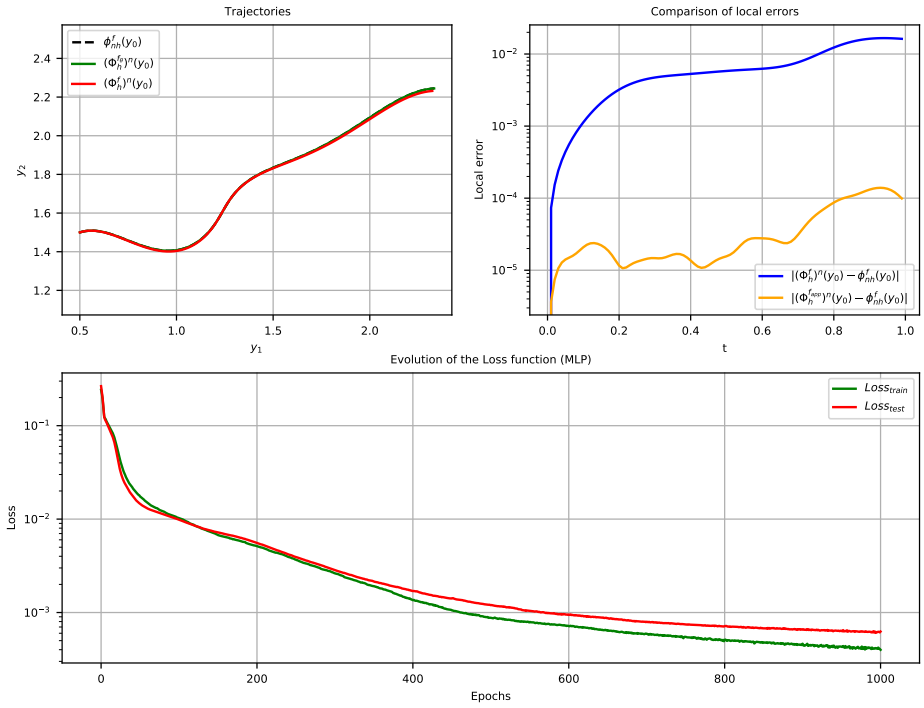

Figure 2.1 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, red: numerical flow with f, green: numerical flow with f andR_θ^ε) and local error (blue: exact flow and numerical flow with f, yellow: exact and numerical flow with f andR_θ^ε) for the inverted pendulum with Integral Forward Euler scheme in the case $\varepsilon = 0.1$.

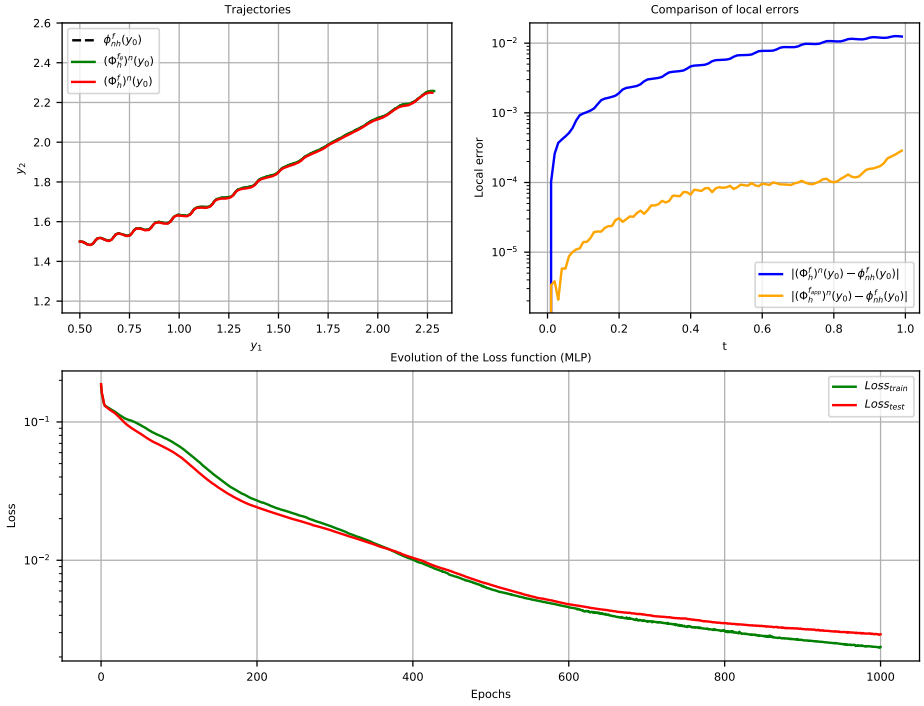

Figure 2.2 – The same comparison in the case $\varepsilon = 0.01$.

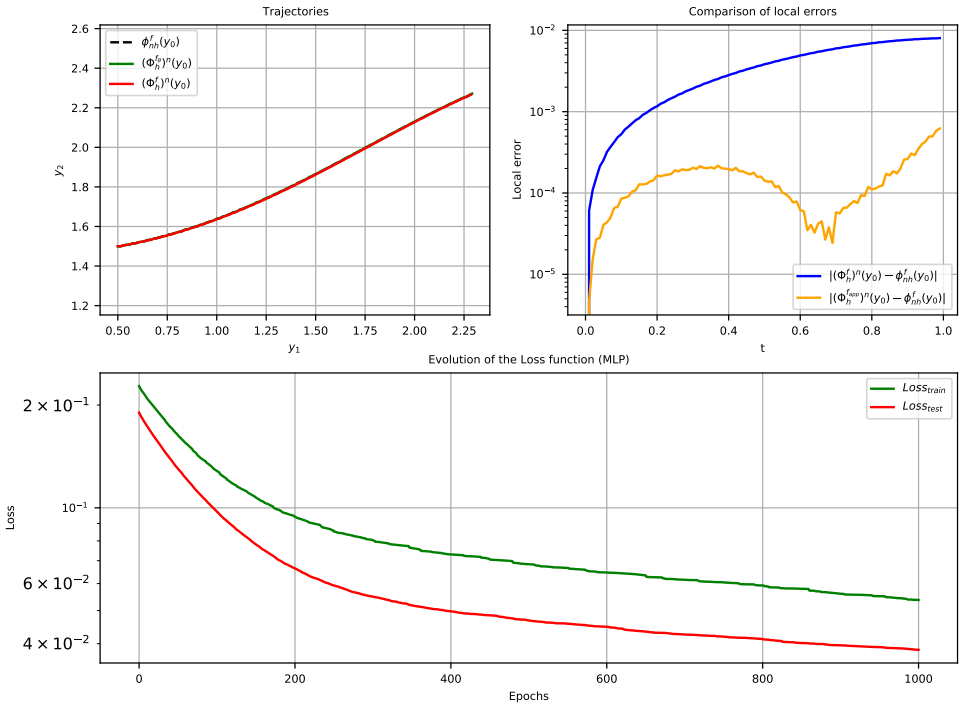

Figure 2.3 – The same comparison in the case $\varepsilon = 0.001$.

Then, we study the global errors between the exact flow and the numerical flow by using the standard Integral Forward Euler scheme, and errors between exact flow and numerical flow by adding the approximated perturbation. Figures 2.4, 2.5 and 2.6 illustrate the estimate obtained in Theorem 27

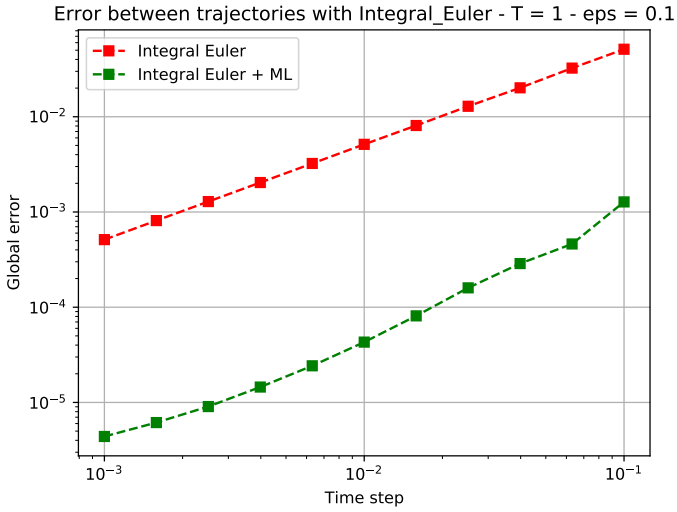

Figure 2.4 – Integration errors (green: integration with f, red: integration with f and R_θ^ε) in the case $\varepsilon = 0.1$

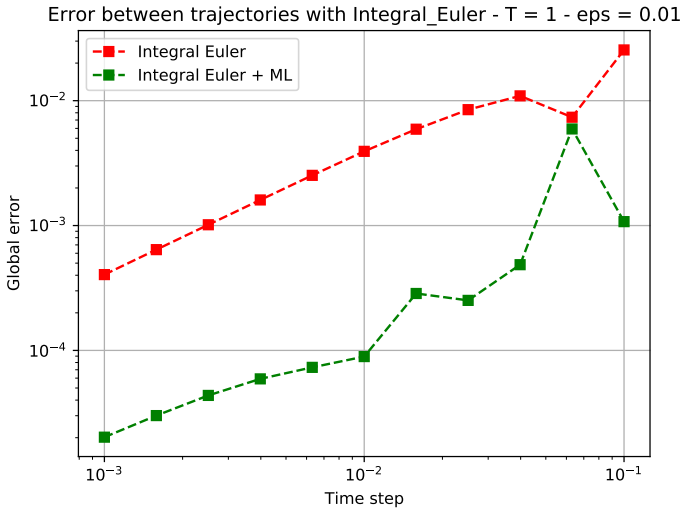

Figure 2.5 – The same comparison in the case $\varepsilon = 0.01$

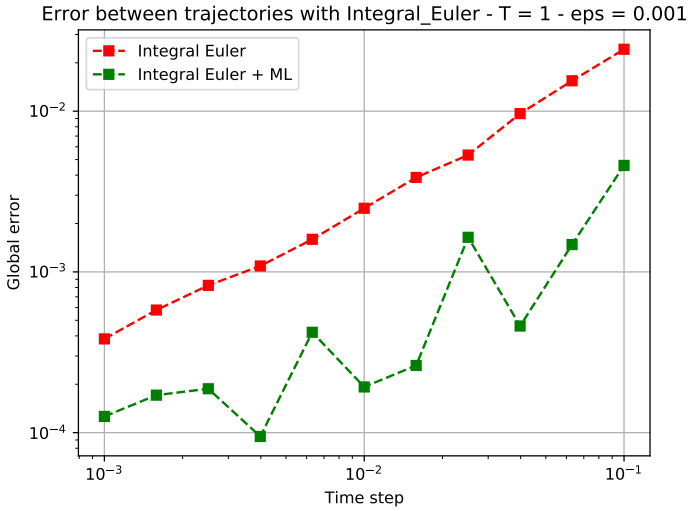

Figure 2.6 – The same comparison in the case $\varepsilon = 0.001$

2.3.2 Computational times

In this subsection, we plot efficiency curves (global error w.r.t. computational time). Three methods are compared: integral forward Euler, integral forward Euler by adding perturbation and integral forward Euler "modified" at the second order given by the scheme (2.6).

Figures 2.7, 2.8 and 2.9 show that performed integral forward Euler method can be more accurate for a fixed computational time, especially for large time steps.

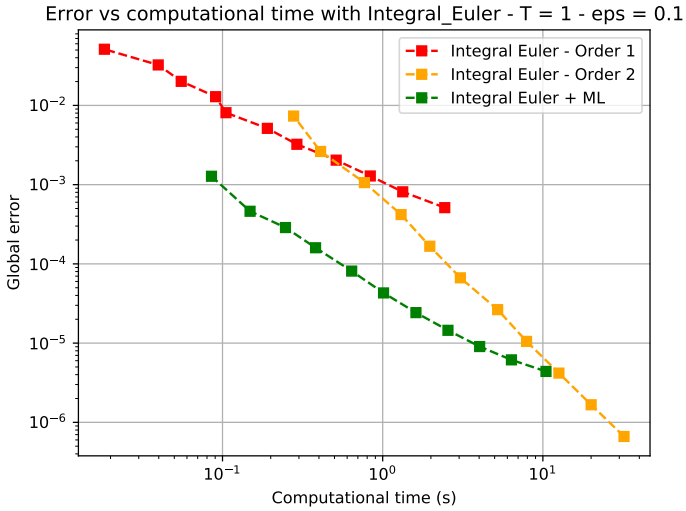

Figure 2.7 – Comparison between computational time and integration error (red: integral forward Euler with f, green: integral forward Euler with f and R_θ^ε, yellow: integration with the second order integral forward Euler) for $\varepsilon = 0.1$.

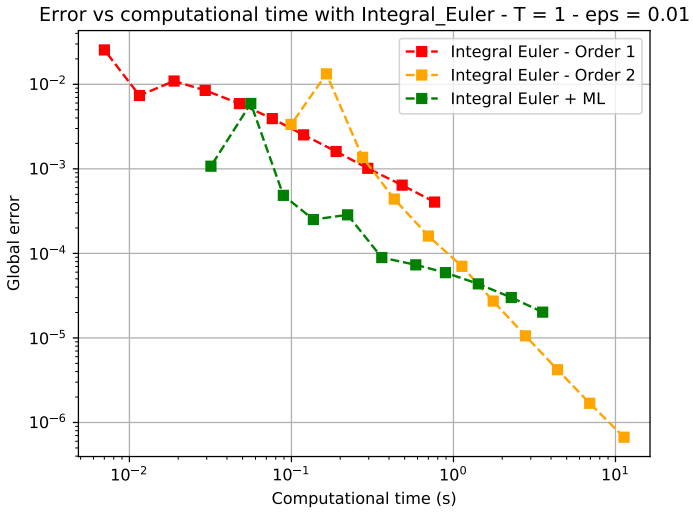

Figure 2.8 – The same simulation in the case $\varepsilon = 0.01$

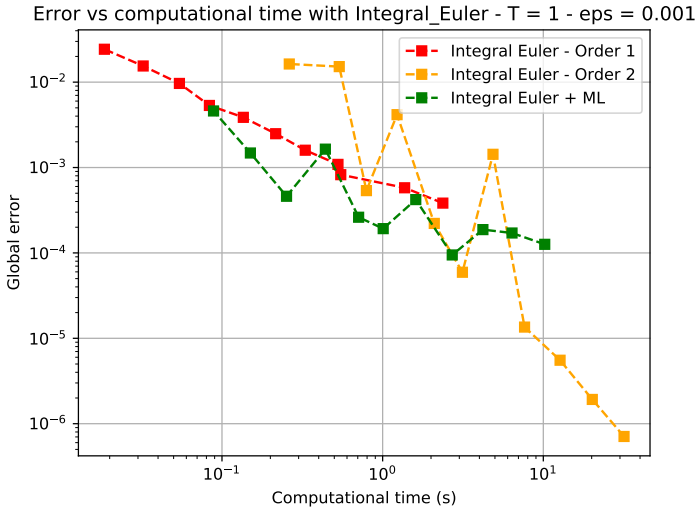

Figure 2.9 – The same simulation in the case $\varepsilon = 0.001$

2.4 Conclusions

In this chapter, an extension of modified equations has been extended to highly oscillatory equations, by using an integral scheme. As in the case of autonomous equations, this scheme can be improved to a more accurate scheme of the same order, able to compete with a second order scheme. However, a good training can require many data, and its number can quickly increase with the dimension. Moreover, a training is available only for one speed of oscillations.

2.A Appendix: Proof of Proposition 26

Proposition 26 can be proved by induction on p.

(i) Base case: If $p = 2$, then we have

$$\begin{aligned} y^\varepsilon(t_0+h) &= y^\varepsilon(t_0) + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y^\varepsilon(t)\right) \mathrm{d}t \\ &= y^\varepsilon(t_0) + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y^\varepsilon(t_0) + \int_{t_0}^{t} f\left(\frac{s}{\varepsilon}, y^\varepsilon(s)\right) \mathrm{d}s\right) \mathrm{d}t. \end{aligned} \tag{2.21}$$

By applying the fundamental theorem of calculus, we get

$$\begin{aligned} y^{\varepsilon}(t_0+h) &= y^{\varepsilon}(t_0) + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y^{\varepsilon}(t_0)\right) \mathrm{d}t \\ &+ \int_{t_0}^{t_0+h} \int_{t_0}^{t} \underbrace{\frac{\partial f}{\partial y}\left(\frac{t}{\varepsilon}, y^{\varepsilon}(s)\right) f\left(\frac{s}{\varepsilon}, y^{\varepsilon}(s)\right)}_{=g_1\left(\frac{t}{\varepsilon}, \frac{s}{\varepsilon}, y^{\varepsilon}(s)\right)} \mathrm{d}s\mathrm{d}t. \end{aligned} \tag{2.22}$$

Thus, we have

$$\begin{aligned} y^{\varepsilon}(t_0+h) &= y^{\varepsilon}(t_0) + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y^{\varepsilon}(t_0)\right) \mathrm{d}t \\ &+ \int_{t_0}^{t_0+h} \int_{t_0}^{t} g_1\left(\frac{t}{\varepsilon}, \frac{s}{\varepsilon}, y^{\varepsilon}(t_0) + \int_{t_0}^{s} f\left(\frac{r}{\varepsilon}, y^{\varepsilon}(r)\right) \mathrm{d}r\right) \mathrm{d}s\mathrm{d}t. \end{aligned} \tag{2.23}$$

If we apply a second time fundamental theorem of calculus, we get

$$\begin{aligned} y^{\varepsilon}(t_0+h) &= y^{\varepsilon}(t_0) + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y^{\varepsilon}(t_0)\right) \mathrm{d}t \\ &+ \int_{t_0}^{t_0+h} \int_{t_0}^{t} \underbrace{\frac{\partial f}{\partial y}\left(\frac{t}{\varepsilon}, y^{\varepsilon}(t_0)\right) f\left(\frac{s}{\varepsilon}, y^{\varepsilon}(t_0)\right)}_{=g_1\left(\frac{t}{\varepsilon}, \frac{s}{\varepsilon}, y^{\varepsilon}(t_0)\right)} \mathrm{d}s\mathrm{d}t. \\ &+ \int_{t_0}^{t_0+h} \int_{t_0}^{t} \int_{t_0}^{s} \frac{\partial g_1}{\partial y}\left(\frac{t}{\varepsilon}, \frac{s}{\varepsilon}, y^{\varepsilon}(r)\right) \mathrm{d}r\mathrm{d}s\mathrm{d}t. \end{aligned} \tag{2.24}$$

which corresponds to the formula in the case $p = 2$.

(ii) Induction step: Assume the formula is true for given $p \in \mathbb{N}, p \geqslant 2$. Thus we have

$$\begin{aligned} y^{\varepsilon}(t_0+h) &= y^{\varepsilon}(t_0) + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y^{\varepsilon}(t_0)\right) \mathrm{d}t \\ &+ \int_{t_0}^{t_0+h} \int_{t_0}^{t} \frac{\partial f}{\partial y}\left(\frac{t}{\varepsilon}, y^{\varepsilon}(t_0)\right) f\left(\frac{s}{\varepsilon}, y^{\varepsilon}(t_0)\right) \mathrm{d}s\mathrm{d}t \\ &+ \sum_{k=2}^{p-1} \int_{t_0}^{t_0+h} \int_{t_0}^{\sigma_0} \cdots \int_{t_0}^{\sigma_{k-1}} g_k\left(\frac{\sigma_0}{\varepsilon}, \frac{\sigma_1}{\varepsilon}, \cdots, \frac{\sigma_k}{\varepsilon}, y^{\varepsilon}(t_0)\right) \mathrm{d}\sigma_k \cdots \mathrm{d}\sigma_1 \mathrm{d}\sigma_0 \\ &+ \int_{t_0}^{t_0+h} \int_{t_0}^{\sigma_0} \cdots \int_{t_0}^{\sigma_{p-1}} g_p\left(\frac{\sigma_0}{\varepsilon}, \frac{\sigma_1}{\varepsilon}, \cdots, \frac{\sigma_p}{\varepsilon}, y^{\varepsilon}(\sigma_p)\right) \mathrm{d}\sigma_p \cdots \mathrm{d}\sigma_1 \mathrm{d}\sigma_0. \end{aligned} \tag{2.25}$$

As we have, by applying fundamental theorem of calculus again:

$$\begin{aligned} g_p\left(\frac{\sigma_0}{\varepsilon},\frac{\sigma_1}{\varepsilon},\cdots,\frac{\sigma_p}{\varepsilon},y^\varepsilon(\sigma_p)\right) &= g_p\left(\frac{\sigma_0}{\varepsilon},\frac{\sigma_1}{\varepsilon},\cdots,\frac{\sigma_p}{\varepsilon},y^\varepsilon(t_0)\right) \\ &+ \int_{t_0}^{\sigma_p} \underbrace{\frac{\partial g_p}{\partial y}\left(\frac{\sigma_0}{\varepsilon},\frac{\sigma_1}{\varepsilon},\cdots,\frac{\sigma_p}{\varepsilon},y^\varepsilon(\sigma_{p+1})\right) f\left(\frac{\sigma_{p+1}}{\varepsilon},y^\varepsilon(\sigma_{p+1})\right)}_{=g_{p+1}\left(\frac{\sigma_0}{\varepsilon},\frac{\sigma_1}{\varepsilon},\cdots,\frac{\sigma_{p+1}}{\varepsilon},y^\varepsilon(\sigma_{p+1})\right)} \mathrm{d}\sigma_{p+1}. \end{aligned} \tag{2.26}$$

Thus, we get:

$$\begin{aligned} y^\varepsilon(t_0+h) &= y^\varepsilon(t_0) + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon},y^\varepsilon(t_0)\right)\mathrm{d}t \\ &+ \int_{t_0}^{t_0+h}\int_{t_0}^{t} \frac{\partial f}{\partial y}\left(\frac{t}{\varepsilon},y^\varepsilon(t_0)\right) f\left(\frac{s}{\varepsilon},y^\varepsilon(t_0)\right)\mathrm{d}s\mathrm{d}t \\ &+ \sum_{k=2}^{p-1}\int_{t_0}^{t_0+h}\int_{t_0}^{\sigma_0}\cdots\int_{t_0}^{\sigma_{k-1}} g_k\left(\frac{\sigma_0}{\varepsilon},\frac{\sigma_1}{\varepsilon},\cdots,\frac{\sigma_k}{\varepsilon},y^\varepsilon(t_0)\right)\mathrm{d}\sigma_k\cdots\mathrm{d}\sigma_1\mathrm{d}\sigma_0 \\ &+ \int_{t_0}^{t_0+h}\int_{t_0}^{\sigma_0}\cdots\int_{t_0}^{\sigma_{p-1}} g_p\left(\frac{\sigma_0}{\varepsilon},\frac{\sigma_1}{\varepsilon},\cdots,\frac{\sigma_p}{\varepsilon},y^\varepsilon(t_0)\right)\mathrm{d}\sigma_p\cdots\mathrm{d}\sigma_1\mathrm{d}\sigma_0 \\ &+ \int_{t_0}^{t_0+h}\int_{t_0}^{\sigma_0}\cdots\int_{t_0}^{\sigma_p} g_{p+1}\left(\frac{\sigma_0}{\varepsilon},\frac{\sigma_1}{\varepsilon},\cdots,\frac{\sigma_{p+1}}{\varepsilon},y^\varepsilon(\sigma_{p+1})\right)\mathrm{d}\sigma_{p+1}\cdots\mathrm{d}\sigma_1\mathrm{d}\sigma_0. \end{aligned} \tag{2.27}$$

$$\begin{aligned} &= y^\varepsilon(t_0) + \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon},y^\varepsilon(t_0)\right)\mathrm{d}t \\ &+ \int_{t_0}^{t_0+h}\int_{t_0}^{t} \frac{\partial f}{\partial y}\left(\frac{t}{\varepsilon},y^\varepsilon(t_0)\right) f\left(\frac{s}{\varepsilon},y^\varepsilon(t_0)\right)\mathrm{d}s\mathrm{d}t \\ &+ \sum_{k=2}^{p}\int_{t_0}^{t_0+h}\int_{t_0}^{\sigma_0}\cdots\int_{t_0}^{\sigma_{k-1}} g_k\left(\frac{\sigma_0}{\varepsilon},\frac{\sigma_1}{\varepsilon},\cdots,\frac{\sigma_k}{\varepsilon},y^\varepsilon(t_0)\right)\mathrm{d}\sigma_k\cdots\mathrm{d}\sigma_1\mathrm{d}\sigma_0 \\ &+ \int_{t_0}^{t_0+h}\int_{t_0}^{\sigma_0}\cdots\int_{t_0}^{\sigma_p} g_{p+1}\left(\frac{\sigma_0}{\varepsilon},\frac{\sigma_1}{\varepsilon},\cdots,\frac{\sigma_{p+1}}{\varepsilon},y^\varepsilon(\sigma_{p+1})\right)\mathrm{d}\sigma_{p+1}\cdots\mathrm{d}\sigma_1\mathrm{d}\sigma_0. \end{aligned} \tag{2.28}$$

So, the proposition is proved.

2.B Appendix: Proof of Theorem 27

Proof the the theorem is similar to the proof of theorem 24 for autonomous equations.

(i) Consistency error: Let introduce the consistency error by

$$\begin{aligned} \varepsilon_{\theta,n} &:= y^{\varepsilon}(t_{n+1}) - y^{\varepsilon}(t_n) - \int_{t_n}^{t_{n+1}} f\left(\frac{t}{\varepsilon}, y^{\varepsilon}(t_n)\right) \mathrm{d}t \\ &- h^2 R_{\theta}^{\varepsilon}\left(\cos\left(\frac{t_n}{\varepsilon}\right), \cos\left(\frac{t_n}{\varepsilon}\right), y^{\varepsilon}(t_n), h\right). \end{aligned} \tag{2.29}$$

Formula (2.8) leads to

$$\begin{aligned} \varepsilon_{\theta,n} &:= h^2 R_{p}^{\varepsilon}\left(\cos\left(\frac{t_n}{\varepsilon}\right), \cos\left(\frac{t_n}{\varepsilon}\right), y^{\varepsilon}(t_n), h\right) \\ &- h^2 R_{\theta}^{\varepsilon}\left(\cos\left(\frac{t_n}{\varepsilon}\right), \cos\left(\frac{t_n}{\varepsilon}\right), y^{\varepsilon}(t_n), h\right) + \mathcal{O}\left(h^{p+1}\right). \end{aligned} \tag{2.30}$$

thus, we get:

$$|\varepsilon_{\theta,n}| \leqslant \delta_{\varepsilon} h^2 + C h^{p+1} \tag{2.31}$$

(ii) Local truncation error: Local truncation error defined with formula (2.17) leads to

$$e_{\theta,n+1} = y_{\theta,n+1}^{\varepsilon} - y^{\varepsilon}(t_{n+1}) \tag{2.32}$$

thanks to definition of the consistency error and formula (2.16), we get

$$\begin{aligned} e_{\theta,n+1} &= e_{\theta,n} + \int_{t_n}^{t_{n+1}} f\left(\frac{t}{\varepsilon}, y_{\theta,n}^{\varepsilon}\right) \mathrm{d}t - \int_{t_n}^{t_{n+1}} f\left(\frac{t}{\varepsilon}, y^{\varepsilon}(t_n)\right) \mathrm{d}t \\ &+ h^2 R_{\theta}^{\varepsilon}\left(\cos\left(\frac{t_n}{\varepsilon}\right), \cos\left(\frac{t_n}{\varepsilon}\right), y_{\theta,n}^{\varepsilon}, h\right) \\ &- h^2 R_{\theta}^{\varepsilon}\left(\cos\left(\frac{t_n}{\varepsilon}\right), \cos\left(\frac{t_n}{\varepsilon}\right), y^{\varepsilon}(t_n), h\right) \\ &- \varepsilon_{\theta,n}. \end{aligned} \tag{2.33}$$

Thus, thanks to estimate over the consistency error, there exist constants L (depending on f) and M (depending on R_{θ}^{ε}, and, as a result, on θ and ε) such that

$$|e_{\theta,n+1}| \leqslant (1 + Lh + Mh^2)|e_{\theta,n}| + \delta_{\varepsilon} h^2 + C h^{p+1} \tag{2.34}$$

Let denote $\tilde{L}_\varepsilon := L + Mh_+$, we get:

$$|e_{\theta,n+1}| \leqslant (1 + \tilde{L}_\varepsilon h)|e_{\theta,n}| + \delta_\varepsilon h^2 + Ch^{p+1}. \tag{2.35}$$

Discrete Grönwall lemma leads to estimate (2.19).

2.C Appendix: Implementations

2.C.1 Numerical integration

In order to compute the integral

$$I(y) = \int_{t_0}^{t_0+h} f\left(\frac{t}{\varepsilon}, y\right) \mathrm{d}t, \tag{2.36}$$

we use the decomposition

$$f(\tau, y) = \langle f \rangle + g(\tau, y) \tag{2.37}$$

where average of g is 0. Thus, we get

$$I(y) = h\langle f \rangle + \int_{t_0}^{t_0+h} g\left(\frac{t}{\varepsilon}, y\right) \mathrm{d}t \tag{2.38}$$

and, after a variable change:

$$I(y) = h\langle f \rangle + \int_{\frac{t_0}{\varepsilon}}^{\frac{t_0+h}{\varepsilon}} g\left(\tau, y\right) \mathrm{d}\tau. \tag{2.39}$$

As $\langle g \rangle = 0$, we have, by denoting $\tau_0 = \frac{t_0}{\varepsilon} + 2\pi\lfloor\frac{t_0}{2\pi\varepsilon}\rceil$ and $\tau_1 = \frac{t_0+h}{\varepsilon}$

$$I(y) = h\langle f \rangle + \int_{\tau_0}^{\tau_1} g\left(\tau, y\right) \mathrm{d}\tau. \tag{2.40}$$

The domain of integration is bounded, so it is easier to do numerical integration. All integrals involved are approximated by Gauss quadrature with 10 points. As a result, computational time is more important for the second-order forward Euler integral scheme, where a double integral is used.

2.C.2 Choice of the parameters for numerical experiments

Parameters	
# Math Parameters:	
Dynamical system:	Pendulum
Numerical method:	Integral Forward Euler
Interval where time steps are selected:	$[h_-, h_+] = [0.001, 0.1]$
Time for ODE simulation:	$T = 1$
Time step for ODE simulation:	$h = 0.01$
High oscillation parameters:	$\varepsilon \in \{0.001, 0.01, 0.1\}$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 10\,000$
Proportion of data for training:	80% - $K_0 = 8\,000$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	10^{-3}
Weight decay:	10^{-9}
Batch size (mini-batching for training):	500
Epochs:	1000
Epochs between two prints of loss value:	100

Computational time for training:

⋆ $\varepsilon = 0.1$: 3 min 16 s

⋆ $\varepsilon = 0.01$: 3 min 16 s

⋆ $\varepsilon = 0.001$: 3 min 15 s

Chapter 3

Deep Learning for Highly oscillatory differential equations

Abstract

Highly oscillatory differential equations, which are often encountered in multi-scale problems, are typically too complex to be solved analytically. However, various numerical methods have been developed to approximate the solutions of these equations. In this paper, we employ averaging theory as an alternative approach to avoid such computations. The strategy involves utilizing neural networks to approximate the vector fields necessary for the pre-computations.

Keywords: Highly oscillatory differential equation, Slow-fast decomposition, Micro-Macro method, auto-encoder, numerical method, averaging theory.

3.1 Introduction

Highly oscillatory differential equations are often used to model phenomena that present a multiscale state with a periodic dependence [25, 21]. Most of these equations cannot be solved analytically. As there are different numerical methods for autonomous differential equations [8, 12, 15, 48, 66, 90], these equations are not suitable for this type of equations due to the stiffness.

Despite the impossibility of using the above methods properly for highly oscillatory ODE's, analytical properties of these differential equations [24, 25, 87] can be used to develop powerful numerical methods. In particular, uniformly accurate

methods are powerful and the error bounds do not depend on the stiffness parameter [21, 22].

These methods require transformations to transform the studied dynamical system into another one in order to apply the numerical methods correctly. For example, preliminary calculations follow the slow-fast decomposition, which separates the dynamics of the multiscale equation into slow and fast dynamics. The micro-macro decomposition doubles the dimension of the dynamical system in order to separate the dynamics and build very accurate numerical methods, while the pullback method is adapted to the geometric properties of the studied dynamical systems.

Despite the accuracy of these methods, preliminary computations are required to transform the dynamical system, and for most of the dynamical systems studied, formal computations are required, which can require a significant computational cost [24, 25].

The main idea of this work is to combine the theory of highly oscillatory differential equations with machine learning techniques and in particular neural networks. Given a highly oscillatory differential equation, the slow-fast decomposition of the equation is first learned by extensive simulations and then approximated by inference from a neural network. Since the slow dynamics is described by an autonomous differential equation, the theory of modified equations [50] is used to learn the slow dynamics. Once this representation of the slow-fast decomposition has been obtained, it is used to solve the equation for any initial value prescribed in a *learnt* range of phase space.

3.1.1 Scope of the paper

The paper is divided into two main sections: Section 3.2 is a presentation of the technique and convergence results, while section 3.3 presents numerical experiments that illustrate the properties of the schemes presented in the previous section.

Subsection 3.2.1 gives the strategy followed to combine machine learning and averaging theory for highly oscillatory differential equations: the main idea is to reproduce the *slow-fast* decomposition using neural networks. It requires simulating exact data with an accurate but time-consuming numerical integrator and using it to train neural networks that learn the decomposition by optimizing a $Loss_{Train}$ error function to learn vector fields associated with the aforementioned decomposition using modified equation theory for slow dynamics and auto-encoder structure for

fast dynamics. An additional method based on the micro-macro decomposition is also presented, which does not require further training with more accurate approximations. Finally, an alternative method adapted to highly oscillatory autonomous systems is presented, which does not require an auto-encoder.

Subsection 3.2.2 establishes an error bound between exact and approximate solutions. The new scheme is not consistent with respect to step size, but it does not require pre-calculations. Compared to the micro-macro based method, better error bounds can be obtained with more uniform accuracy. Finally, error bounds are given for alternative methods, similar to the first method.

For the numerical experiments, forward Euler and midpoint schemes are used to illustrate the main contribution of this work: the correct learning of both vector fields required for the slow-fast decomposition and the possibility to accurately approximate solutions of highly oscillatory equations without making preliminary computations before numerical integration by using *slow-fast* decomposition or Micro-Macro based methods, and for autonomous case, numerical experiments show a comparison with the classical method based on *slow-fast* decomposition.

3.1.2 Related work

Links between differential equations and machine learning on the one hand, and multiscale problems on the other hand, have already been explored in several publications. The link between differential equations and machine learning is mainly done by learning the hidden dynamics from collected data, using regression [11, 36] or statistical methods [81, 92], while the link between multi-scale problems is mainly done by learning solutions [60].

Modified equation coupling. Link with modified equation theory has been used to study the learning of the hidden dynamics of a differential equation [36, 117]. Offen et al. used it to learn the Hamiltonian function [85].

Neural network structure and geometric properties. To preserve the properties of the equation, different neural network structures have been used. To preserve asymptotic properties for multi-scale equations, Jin et al. have used Physics Informed Neural Networks (PINN's) [60]. In addition, other structures can be developed to study multiscale equations [67, 74, 109], such as Convolutional Neural Networks (CNN's) [114]. To preserve geometric properties, such as the Hamiltonian vector field, Hamiltonian Neural Networks have been used [32, 44]. Finally, to learn

reciprocal mappings, auto-encoders are used to accomplish this task, as has been done by Jin et al. to learn Poisson systems [58].

Approximation by Neural Networks. In order to properly approximate functions by neural networks, error estimates have been established. Anastassiou [1, 2] gave convergence rates for approximations of functions taking values in finite and infinite dimensional vector spaces by networks, depending on the number of parameters and the dimension of the input. By considering neural network spaces as functional spaces, it is possible to give error bounds as a function of the number of parameters [33, 45]. In a separate work, Bach [4] has given estimates of the approximation error by considering neural networks as elements of a Hilbertian space. Finally, an approximation problem [14] that is highlighted is the curse of dimensionality, where high-dimensional vector fields are approximated with a slower rate of convergence than low-dimensional vector fields, i.e. for high dimensions, more parameters and more data are required to obtain satisfactory learning.

3.2 Approximate solutions of highly oscillatory differential equations with machine learning

Consider an highly oscillatory differential equation of the form

$$\begin{cases} \dot{y^\varepsilon}(t) &= f\left(\frac{t}{\varepsilon}, y^\varepsilon(t)\right) \in \mathbb{R}^d, \quad t \in [0,T] \\ y^\varepsilon(0) &= y_0 \end{cases} \tag{3.1}$$

where $f : \mathbb{R} \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ is assumed to be sufficiently smooth and 2π-periodic with respect to its first variable. Additionally, the parameter $\varepsilon \in (0,1]$ introduces high oscillations. By the Cauchy-Lipschitz theorem, we are guaranteed the existence and uniqueness of a solution for any initial value $y_0 \in \mathbb{R}^d$. Our objective is to approximate the solution over the interval $[0,T]$ at times $t_n = nh$, where $0 \leqslant n \leqslant N$, with $h = \frac{T}{N}$ being the time step and N the number of discretization points.

3.2.1 General strategy

As explained in the Introduction section, we will approximate the *slow-fast* decomposition using neural networks to approximate the solution of (3.1), particularly

when the parameter ε is small. Furthermore, additional methods based on the micro-macro scheme are employed to obtain more accurate approximations.

3.2.1.1 Slow-fast decomposition

Let us consider $t \mapsto \varphi_t^f(y_0)$, the exact flow associated with the equation (3.1). We aim to perform the following decomposition:

$$\varphi_t^f(y_0) = \phi^\varepsilon_{\frac{t}{\varepsilon}}\left(\psi_t^\varepsilon(y_0)\right) \tag{3.2}$$

where $\tau \mapsto \phi_\tau^\varepsilon$ is 2π-periodic, $\phi_0^\varepsilon = \mathrm{Id}$ [1] and $\phi^\varepsilon = \mathrm{Id} + \mathcal{O}(\varepsilon)$. ϕ^ε introduces high oscillations while $t \mapsto \psi_t^\varepsilon$ results in global drift. $t \mapsto \psi_t^\varepsilon(y_0)$ is the solution of an autonomous differential equation with an associated vector field, denoted F^ε, called the *averaged field*. ϕ^ε and F^ε are represented by the following formal series with respect to ε:

$$\phi_\tau^\varepsilon(y) = y + \sum_{j=1}^{+\infty} \varepsilon^j \tilde{\phi}_j(\tau, y) \quad \text{and} \quad F^\varepsilon(y) = \langle f \rangle(y) + \sum_{j=1}^{+\infty} \varepsilon^j F_j(y), \tag{3.3}$$

where $\langle f \rangle$ is the *average field* defined as

$$\langle f \rangle(y) := \frac{1}{2\pi} \int_0^{2\pi} f(\tau, y)\, d\tau, \tag{3.4}$$

and the coefficient functions ϕ_j and F_j depend on derivatives of f and $\langle f \rangle$. However, the formula (3.2) may not hold due to the fact that ϕ^ε and F^ε are defined by formal series that generally do not converge. Despite the non-convergence of the formal series (3.3), one can describe an approximation of the slow-fast decomposition. Specifically, for all $n \in \mathbb{N}$, let $F^{\varepsilon,[n]}$ and $\phi^{\varepsilon,[n]}$ denote the truncation of ϕ^ε and F^ε by neglecting the $\mathcal{O}(\varepsilon^{n+1})$ terms. According to averaging theory [24, 26], there exists $\varepsilon_0 > 0$ such that, for all $\varepsilon \in (0, \varepsilon_0]$, there exists $n_\varepsilon \in \mathbb{N}$ such that, for all $t \in [0, T]$,

$$\left| \varphi_t^f(y_0) - \phi^{\varepsilon,[n_\varepsilon]}_{\frac{t}{\varepsilon}}\left(\varphi_t^{F^{\varepsilon,[n_\varepsilon]}}(y_0)\right) \right| \leqslant M e^{-\frac{\beta\varepsilon_0}{\varepsilon}} \tag{3.5}$$

for some constants $M, \beta > 0$ independent from ε. Since the error bound decays very rapidly as $\varepsilon \to 0$, the slow-fast decomposition serves as a highly accurate approximation of the solution.

1. The hypothesis $\phi_0^\varepsilon = \mathrm{Id}$ arises from stroboscopic averaging [21]. An alternative convention, standard averaging, considers the average $\langle \phi_\cdot^\varepsilon \rangle = \mathrm{Id}$.

3.2.1.2 Micro-Macro decomposition

Despite the efficiency of the *slow-fast* decomposition for small parameters ε, it is not a uniformly accurate approximation of the solution with respect to ε due to the exponential error (3.5). To correct this error, the micro-macro decomposition [21] exploits the multiscale structure of the equation and provides a decomposition of the solution into a *slow-fast* part and a remainder. Considering $\phi^{[p]}$ and $F^{[p]}$ as truncations of ϕ^{ε} and F^{ε} by neglecting $\mathcal{O}(\varepsilon^{p+1})$ terms, we obtain, for all $t \in [0, T]$, the following:

$$y^{\varepsilon}(t) = \phi^{[p]}(v(t)) + w(t), \tag{3.6}$$

where (v, w) is the solution of the micro-macro system:

$$\left\{ \begin{array}{rcl} \dot{v}(t) & = & F^{[p]}(v(t)) \\ \dot{w}(t) & = & f\left(\frac{t}{\varepsilon}, \phi^{[p]}(v(t)) + w(t)\right) - \frac{1}{\varepsilon}\frac{\partial}{\partial \tau}\phi^{[p]}_{\frac{t}{\varepsilon}}(v(t)) - \frac{\partial}{\partial y}\phi^{[p]}_{\frac{t}{\varepsilon}}(v(t))F^{[p]}(v(t)) \end{array} \right. , \tag{3.7}$$

with $(v, w)(0) = (y^{\varepsilon}(0), 0)$. Using a numerical integrator of order p for this system yields an approximate solution of order p, $(v_n, w_n)_{n\in\mathbb{N}}$, with respect to the step size h:

$$\max_{0\leqslant n\leqslant N} \left|\phi^{[p]}(v(t_n)) + w(t_n) - \phi^{[p]}(v_n) - w_n\right| \leqslant \overline{M} h^p, \tag{3.8}$$

where the constant $\overline{M}$ does not depend on ε. This represents a uniformly accurate approximation, and the micro-macro method is considered a *Uniformly Accurate* (UA) method.

3.2.1.3 Machine learning method

The main idea of this paper is to approximate the map $(\tau, y, \varepsilon) \mapsto \phi_{\tau}^{\varepsilon,[n_\varepsilon]}(y)$ and the flow $(t, y, \varepsilon) \mapsto \varphi_t^{F^{\varepsilon},[n_\varepsilon]}(y)$ using neural networks to achieve a similar structure for the approximated solution as (3.2).

Since the flow $(t, y, \varepsilon) \mapsto \varphi_t^{F^{\varepsilon},[n_\varepsilon]}(y)$ is associated with an autonomous differential equation, backward error analysis (modified equation theory [50]) can be employed to approximate solutions of this equation [10]. If we use a numerical method of order

p and a given time step Φ_h, then by denoting $\widetilde{F_h^{\varepsilon,[n_\varepsilon]}}$ as the modified field $F^{\varepsilon,[n_\varepsilon]}$ with respect to Φ_h and $\widetilde{F_h^{\varepsilon,[n_\varepsilon],[q]}}$ as the truncation of $\widetilde{F_h^{\varepsilon,[n_\varepsilon]}}$ at order h^{p+q-1} [10, 50], we have, for all compact set $\mathbb{K} \subset \mathbb{R}^d$ and $y \in \mathbb{K}$:

$$\left| \varphi_h^{F^{\varepsilon,[n_\varepsilon]}}(y) - \Phi_h^{\widetilde{F_h^{\varepsilon,[n_\varepsilon],[q]}}}(y) \right| \leqslant \overline{C} h^{p+q-1}. \tag{3.9}$$

Then we need to approximate $\widetilde{F_h^{\varepsilon,[n_\varepsilon],[q]}}$ with a neural network, called F_θ, which can be interpreted as a perturbation of the average field:

$$F_\theta(y, h, \varepsilon) = \langle f \rangle + R_{\theta,F}(y, h, \varepsilon) \tag{3.10}$$

where the perturbation $R_{\theta,F}$ is a multilayer perceptron (MLP).

Furthermore, we must account for the high oscillation generator described by the map $(\tau, y, \varepsilon) \mapsto \phi_\tau^\varepsilon(y)$. To access the autonomous equation from the solution of the initial equation, we need to approximate both $\phi_\tau^{\varepsilon,[n_\varepsilon]}$ and its inverse [2]. Therefore, an auto-encoder consisting of a pair of neural networks $(\phi_\theta(\tau, y, \varepsilon), \phi_{\theta,-}(\tau, y, \varepsilon))$ must be used to approximate $\left(\phi_\tau^{\varepsilon,[n_\varepsilon]}(y), (\phi_\tau^{\varepsilon,[n_\varepsilon]})^{-1}(y)\right)$. In order to preserve the structure of $\phi^{\varepsilon,[n_\varepsilon]}$ and its inverse, as required by stroboscopic averaging, both neural networks can be considered as perturbations of the identity:

$$\phi_\theta(\tau, y, \varepsilon) = y + \varepsilon \left[R_{\theta,+}\left(\cos(\tau), \sin(\tau), y, \varepsilon\right) - R_{\theta,+}\left(1, 0, y, \varepsilon\right) \right] \tag{3.11}$$

and

$$\phi_{\theta,-}(\tau, y, \varepsilon) = y + \varepsilon \left[R_{\theta,-}\left(\cos(\tau), \sin(\tau), y, \varepsilon\right) - R_{\theta,-}\left(1, 0, y, \varepsilon\right) \right] \tag{3.12}$$

where both $R_{\theta,+}$ and $R_{\theta,-}$ are multilayer perceptrons. Furthermore, both mappings $y \longmapsto \phi_\theta(\tau, \cdot, \varepsilon) \circ \phi_{\theta,-}(\tau, \cdot, \varepsilon)$ and $y \longmapsto \phi_{\theta,-}(\tau, \cdot, \varepsilon) \circ \phi_\theta(\tau, \cdot, \varepsilon)$ must be close to identity to match the auto-encoder structure [58].

The complete numerical procedure can be divided into three main steps: First, data are collected by simulating the exact flow very accurately at different points in

2. this map is invertible if ε is small enough, as a perturbation of the identity

the domain. A high number of simulations and high accuracy are prerequisites for a good approximation of the averaged field and a high oscillation generator. Secondly, the different neural networks are trained separately by minimizing a prescribed loss function. Finally, given initial data, an approximation to the exact solution is obtained by applying the same numerical scheme to the neural networks as was used for training. In more detail on the three stages:

1. **Construction of the data set:** K "initial" data y_0^k at time t_0^k are randomly selected into a compact set $\Omega \subset \mathbb{R}^d$ (where we want to simulate the solution) with uniform distribution. The initial time t_0^k is randomly chosen in $[0, 2\pi]$. Then, for all $0 \leqslant k \leqslant K-1$, we compute a very accurate approximation of the exact flow at times h^k with initial condition y_0^k, denoted y_1^k. Time steps h^k and high oscillation parameters ε^k are chosen in domains $[h_-, h_+]$ and $[\varepsilon_-, \varepsilon_+]$ respectively (we actually pick the values $\log h^k$ and $\log \varepsilon^k$ randomly in the domains $[\log h_-, \log h_+]$ and $[\log \varepsilon_-, \log \varepsilon_+]$ with uniform distribution).
2. **Training the neural networks:** We minimize the Mean Squared Error (MSE), denoted $Loss_{Train}$, which measures the difference between the predicted data $\hat{y}_1^k$ and the "exact data" y_1^k, by computing the optimal parameters of the NN over K_0 data (where $1 \leqslant K_0 \leqslant K-1$) using a gradient method:

$$
\begin{aligned}
Loss_{Train} \;=\; & \frac{1}{K_0}\sum_{k=0}^{K_0-1} \Bigg| \underbrace{\phi_\theta\Big(\frac{t_0^k+h^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)\circ \Phi_{h^k}^{F_\theta(\cdot,h^k,\varepsilon^k)} \circ \phi_{\theta,-}\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)\Big(y_0^k\Big)}_{=\hat{y_1}^k} \\
& - \underbrace{\varphi^f_{t_0^k,h^k}\Big(y_0^k\Big)}_{=y_1^k}\Bigg|^2 \\
& + \frac{1}{K_0}\sum_{k=0}^{K_0-1} \Big|\phi_\theta\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)\circ \phi_{\theta,-}\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)\Big(y_0^k\Big) - y_0^k\Big|^2 \qquad (3.13) \\
& + \frac{1}{K_0}\sum_{k=0}^{K_0-1} \Big|\phi_{\theta,-}\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)\circ \phi_\theta\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)\Big(y_0^k\Big) - y_0^k\Big|^2
\end{aligned}
$$

Note that the first term of $Loss_{Train}$ corresponds to the structure of the slow-fast decomposition equation, while the second and third terms correspond to the auto-encoder structure for the pair $(\phi_\theta, \phi_{\theta,-})$.

At the same time, we compute the value of another MSE, called $Loss_{Test}$,

which measures the difference between the predicted data $\hat{y}_1^k$ and the "exact data" y_1^k for a subset of the initial values that were not used to train the NNs. The goal of this step is to estimate the performance of the training for "unknown" initial values:

$$\begin{aligned} Loss_{Train} &= \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\Big|\underbrace{\phi_\theta\Big(\frac{t_0^k+h^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)\circ\Phi_{h^k}^{F_\theta(\cdot,h^k,\varepsilon^k)}\circ\phi_{\theta,-}\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)(y_0^k)}_{=\hat{y_1}^k} \\ &- \underbrace{\varphi^f_{t_0^k,h^k}(y_0^k)}_{=y_1^k}\Big|^2 \\ &+ \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\Big|\phi_\theta\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)\circ\phi_{\theta,-}\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)(y_0^k)-y_0^k\Big|^2 \\ &+ \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\Big|\phi_{\theta,-}\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)\circ\phi_\theta\Big(\frac{t_0^k}{\varepsilon^k},\cdot,\varepsilon^k\Big)(y_0^k)-y_0^k\Big|^2 \end{aligned} \tag{3.14}$$

If $Loss_{Train}$ and $Loss_{Test}$ show the same decay pattern, this indicates that there is no overfitting, i.e. the neural network model does not fit exactly to its training data and remains able to perform accurately on unseen data, which is its main purpose.

3. **Numerical approximation using *slow-fast* decomposition:** At the end of training, an accurate approximation $F_\theta(\cdot,h,\varepsilon)$ of $\widetilde{F_h^{\varepsilon,[n_\varepsilon],[q]}}$ is available. It is then used to compute the successive values of $\left(\Phi_h^{F_\theta(\cdot,h,\varepsilon)}\right)^n(y_0)$ for $n=0,\ldots,N$. There is also an exact approximation $\phi_\theta(\cdot,\cdot,\varepsilon)$ of ϕ^ε. So an approximation of the solution can be computed by plotting:

$$y_{\theta,n}^\varepsilon := \phi_\theta\left(\frac{t_n}{\varepsilon},\left(\Phi_h^{F_\theta(\cdot,h,\varepsilon)}\right)^n(y_0),\varepsilon\right) \tag{3.15}$$

for all $n=0,\cdots,N$.

4. **Numerical approximation using micro-macro correction:** At the end of the training, we have the alternative choice to reproduce the numerical integration based on Micro-Macro by plotting

$$y_{\theta,n}^\varepsilon = \phi_\theta\left(\frac{t_n}{\varepsilon},v_{\theta,n},\varepsilon\right)+w_{\theta,n}, \tag{3.16}$$

where, for all $n \in \mathbb{N}$:

$$v_{\theta,n} = \left(\Phi_{t_n,h}^{F_\theta(\cdot,h,\varepsilon)}\right)^n (y_0), \tag{3.17}$$

and

$$w_{\theta,n+1} = \left(\Phi_{t_n,h}^{g_\theta(\frac{t_n}{\varepsilon},\cdot,v_{\theta,n},\varepsilon)}\right)(w_{\theta,n}), \tag{3.18}$$

where, for all $(\tau, w, v, \varepsilon) \in [0, 2\pi] \times \Omega \times \Omega \times]0, 1]$,

$$\begin{aligned} g_\theta(\tau, w, v, \varepsilon) &= f(\tau, \phi_\theta(\tau, v, \varepsilon) + w) - \frac{1}{\varepsilon}\partial_\tau \phi_\theta(\tau, v, \varepsilon) \\ &- \partial_y \phi_\theta(\tau, v, \varepsilon) F_\theta(v, 0, \varepsilon). \end{aligned} \tag{3.19}$$

The micro-macro correction has the advantage of avoiding learning the whole micro-macro vector field with the double dimension of the data set, which requires more data. Moreover, this method does not require more training than the *slow-fast* decomposition-based method.

3.2.1.4 An alternative method for autonomous systems

Let us consider now autonomous highly oscillatory systems of the form

$$\dot{y^\varepsilon}(t) = \frac{1}{\varepsilon} A y^\varepsilon(t) + g(y^\varepsilon(t)), \tag{3.20}$$

where $A \in \mathcal{M}_d(\mathbb{R})$ with eigenvalues in $i\mathbb{Z}$ and $g : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ is a smooth function. A first way to study this system is to change the function $z^\varepsilon(t) = e^{-\frac{t}{\varepsilon}A} y^\varepsilon(t)$ to get the new system.

$$\dot{z^\varepsilon}(t) = e^{-\frac{t}{\varepsilon}A} g\left(e^{\frac{t}{\varepsilon}A} z^\varepsilon(t)\right), \tag{3.21}$$

which is a system of the form (3.1).

However, this system can be studied directly in its autonomous form. A normal form theorem [19] says that there exists a matrix A^ε and a vector field g^ε such that A^ε generates a periodic flow $\tau \longmapsto \Phi_\tau^\varepsilon$, g^ε generates a flow $t \longmapsto \varphi_t^{g^\varepsilon}$, the Lie bracket

$[A^\varepsilon, g^\varepsilon]$ vanishes [3] and for all $T > 0$ we can find a positive constant C_T, which we check for all $t \in [0, T]$ and $\varepsilon \in]0, 1]$:

$$\left| y^\varepsilon(t) - \phi^\varepsilon_{\frac{t}{\varepsilon}} \left(\varphi^{g^\varepsilon}_t (y^\varepsilon(0)) \right) \right| \leqslant C_T e^{-\frac{C_T}{\varepsilon}}. \tag{3.22}$$

The main idea here is to approximate the flow ϕ^ε and φ^{g^ε} with neural networks. This method has the advantage of not requiring an auto-encoder to learn ϕ^ε.

So we approximate $\varphi^{g^\varepsilon}_h$ with a neural network called φ_θ as an identity perturbation:

$$\varphi_\theta(y, h, \varepsilon) = y + hR_{\theta,\varphi}(y, h, \varepsilon), \tag{3.23}$$

where $R_{\theta,\varphi}$ is a multilayer perceptron.

We also approximate the periodic flow $\tau \longmapsto \phi^\varepsilon_\tau$ by using a neural network called ϕ_θ as an identity perturbation:

$$\phi_\theta(\tau, y, \varepsilon) = y + \left[R_{\theta,\phi}(\cos(\tau), \sin(\tau), y, \varepsilon) - R_{\theta,\phi}(1, 0, y, \varepsilon) \right]. \tag{3.24}$$

where $R_{\theta,\phi}$ is also a multilayer perceptron.

As in the classical case, the numerical procedure is also divided into three main steps: Data generation, training of the neural networks by *Loss* minimisation and numerical integration. More precisely, we follow the same procedure as in the classical case:

1. **Data set construction:** K initial data y^k_0 at time $t = 0$ are randomly selected into a compact set $\Omega \subset \mathbb{R}^d$ with uniform distribution. We randomly select $h^k \in [h_-, h_+]$ and $\varepsilon^k \in [\varepsilon_-, \varepsilon_+]$ (where $\log h^k$ and $\log \varepsilon^k$ are randomly chosen with uniform distribution). For all $0 \leqslant k \leqslant K - 1$, we compute $y^k_1 = y^\varepsilon(h)$ using an accurate (and expensive) integrator (very accurate approximation of the exact flow).
2. **Training the neural networks:** We minimize the Mean Squared Error (MSE) $Loss_{Train}$ function, which measures the difference between the "exact"

3. A^ε can be considered as a linear vector field $y \longmapsto A^\varepsilon y$. So we have $[A^\varepsilon, g^\varepsilon](y) = A^\varepsilon g^\varepsilon(y) - \partial_y g^\varepsilon(y) A^\varepsilon(y)$

and predicted data, using a gradient method to find optimal NN parameters over K_0 data:

$$\begin{aligned} Loss_{Train} &:= \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|y_1^k-\phi_\theta\left(\frac{h^k}{\varepsilon^k},\varphi_\theta(y_0^k,h^k,\varepsilon^k),\varepsilon^k\right)\right|^2 \\ &+ \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|\phi_\theta\left(\frac{h^k}{\varepsilon^k},\varphi_\theta(y_0^k,h^k,\varepsilon^k),\varepsilon^k\right)\right. \\ &- \left.\varphi_\theta\left(\phi_\theta\left(\frac{h^k}{\varepsilon^k},y_0^k,\varepsilon^k\right),h^k,\varepsilon^k\right)\right|^2. \end{aligned} \tag{3.25}$$

Notice that the first term of $Loss_{Train}$ corresponds to the structure of the equation with both flows, whereas the second term corresponds to the property of flow commutativity (which is equivalent to the vanishing of the Lie bracket of associated vector fields).

At the same time, we compute the MSE $Loss_{Test}$ in order to measure the performance of the training for "unknown" data:

$$\begin{aligned} Loss_{Test} &:= \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\left|y_1^k-\phi_\theta\left(\frac{h^k}{\varepsilon^k},\varphi_\theta(y_0^k,h^k,\varepsilon^k),\varepsilon^k\right)\right|^2 \\ &+ \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\left|\phi_\theta\left(\frac{h^k}{\varepsilon^k},\varphi_\theta(y_0^k,h^k,\varepsilon^k),\varepsilon^k\right)\right. \\ &- \left.\varphi_\theta\left(\phi_\theta\left(\frac{h^k}{\varepsilon^k},y_0^k,\varepsilon^k\right),h^k,\varepsilon^k\right)\right|^2. \end{aligned} \tag{3.26}$$

3. **Integration:** At the end of the training, we have an accurate approximation of ϕ^ε and φ^{g^ε}. So we plot the points $(y^\varepsilon_{\theta,n})_{0\leqslant n\leqslant N} = \left(\phi_\theta\left(\frac{t_n}{\varepsilon},\varphi_\theta(\cdot,h,\varepsilon)^n(y^\varepsilon(0)),\varepsilon\right)\right)_{0\leqslant n\leqslant N}$ for all $n=0,\cdots,N$.

3.2.2 Error analysis

In this subsection we analyse the error resulting from the methods described in the previous subsection. More specifically, we give estimates of the global error for each standard numerical method.

3.2.2.1 Slow-fast decomposition

Slow-fast decomposition provides a direct numerical method with corresponding error bounds.

Theorem 28. *Let denote the following learning errors:*

(i) Learning error for high oscillation generator:

$$\delta_\phi := \left|\left| \frac{\phi^{\varepsilon,[n_\varepsilon]} - \phi_\theta}{\varepsilon} \right|\right|_{L^\infty([0,2\pi]\times\Omega\times[0,\varepsilon_+])} \tag{3.27}$$

(ii) Learning error for modified averaged field:

$$\delta_F := \left|\left| \widetilde{F_h^{\varepsilon,[n_\varepsilon],[q]}} - F_\theta \right|\right|_{L^\infty(\Omega\times[0,h_+]\times[0,\varepsilon_+])}, \tag{3.28}$$

Let consider these two hypotheses:

(i) *For $f_1, f_2 : \Omega \to \mathbb{R}^d$ smooth enough, we have, for all $h > 0$,*

$$\left|\left|\Phi_h^{f_1} - \Phi_h^{f_2}\right|\right|_{L^\infty(\Omega)} \leqslant Ch\,||f_1 - f_2||_{L^\infty(\Omega)} \tag{3.29}$$

for some positive constant $C > 0$ independent from f_1 and f_2.

(ii) *For $f : \Omega \to \mathbb{R}^d$ smooth enough, there exists $L_f > 0$ s.t. for all $y1, y2 \in \Omega$, for all $h > 0$, we have*

$$\left|\Phi_h^f(y_1) - \Phi_h^f(y_2)\right| \leqslant (1 + L_f h)|y1 - y2| \tag{3.30}$$

Let denote $y_{\theta,n}^\varepsilon$ the following numerical flow:

$$y_{\theta,n}^\varepsilon := \phi_\theta\left(\frac{t_n}{\varepsilon}, \left(\Phi_h^{F_\theta(\cdot,h,\varepsilon)}\right)^n (y_0), \varepsilon\right). \tag{3.31}$$

Then there exist constants $\lambda, \alpha > 0$ (independent from h, ε) s.t. for all $h \leqslant h_+$ and $\varepsilon \in]0, \varepsilon_0]$:

$$\underset{0\leqslant n\leqslant N}{Max} \left|y^\varepsilon(t_n) - y_{\theta,n}^\varepsilon\right| \leqslant Me^{-\frac{\beta\varepsilon_0}{\varepsilon}} + \delta_\phi\varepsilon + (1+\alpha\varepsilon)\frac{e^{\lambda T} - 1}{\lambda}\left[\overline{C}h^{p+q-1} + C\delta_F\right] \tag{3.32}$$

Remarks. ***(i)*** *The vector fields $\phi_\cdot^{\varepsilon,[n_\varepsilon]}$ and $\widetilde{F_h^{\varepsilon,[n_\varepsilon]}}$ are smooth by construction (in the formulas* (3.51) *and* (3.52)*). As for $\phi_{\theta,-}$, ϕ_θ, and F_θ, they are also smooth since they are obtained by the composition of the affine function $A_1, \cdots, A_{L+1}$ and non-linear functions $\sigma_1, \cdots, \sigma_L$ (the so-called activation functions). For L layers, the output of the neural network appears to be of the form $A_{L+1} \circ \Sigma_L \circ A_L \circ \cdots \Sigma_1 \circ A_1$. So if the activation functions are smooth, then so are $\phi_{\theta,-}$, ϕ_θ, and F_θ. This is the case, for example, when Σ_i are the hyperbolic tangent functions.*

(ii) *A similar error estimate holds for a variable step size implementation of the numerical method Φ: if we actually use the step sequence $0 \leq h_j \leq h_+$, then $T = h_0 + \cdots + h_{N-1}$, $t_n = h_0 + \cdots + h_{n-1}$ and*

$$y_{\theta,n} = \phi_\theta\left(\frac{t_n}{\varepsilon}, \Phi_{h_{n-1}}^{F_\theta(\cdot,h_{n-1},\varepsilon)} \circ \ldots \circ \Phi_{h_0}^{F_\theta(\cdot,h_0,\varepsilon)}(y^\varepsilon(0)), \varepsilon\right) \tag{3.33}$$

constants $\lambda, \alpha > 0$ (independent from h, ε) such that for all $h \leq h$ and $\varepsilon \in]0,1]$:

$$\underset{0\leqslant n\leqslant N}{Max} |y^\varepsilon(t_n) - y_{\theta,n}| \leqslant Me^{-\frac{\beta\varepsilon_0}{\varepsilon}} + \delta_\phi\varepsilon + (1+\alpha\varepsilon)\frac{e^{\lambda T}-1}{\lambda}\left[\overline{C}h^{p+q-1} + C\delta_F\right]. \tag{3.34}$$

3.2.2.2 Micro-Macro correction

By employing the method of *slow-fast* decomposition to derive a numerical technique based on micro-macro decomposition, we arrive at this novel numerical method along with its corresponding error bounds for numerical approximation.

Theorem 29. *Let consider $p \in \mathbb{N}^*$ and denote the following learning errors:*

(i) Learning error for high oscillation generator:

$$\delta_\phi := \left|\left|\frac{\phi^{[p]} - \phi_\theta}{\varepsilon}\right|\right|_{W^{1,\infty}(\Omega\times[0,2\pi]),L^\infty([0,\varepsilon_+])} \tag{3.35}$$

(ii) Learning error for modified averaged field:

$$\delta_F := \left|\left|\widetilde{F_h^{[p],[q]}} - F_\theta\right|\right|_{L^\infty(\Omega\times[0,h_+]\times[0,\varepsilon_+])} \tag{3.36}$$

(iii) Learning error for field of associated to micro part:

$$\delta_g := ||g - g_\theta||_{L^\infty([0,2\pi]\times\Omega\times\Omega\times[0,\varepsilon_+])} \tag{3.37}$$

Let us denote $\Phi_{t,h}$ a numerical method of order p (depending on time t). Let consider these two hypotheses:

(i) *For $f_1, f_2 : [0,T]\times\Omega \to \mathbb{R}^d$ smooth enough and $t \in [0,T]$, we have, for all $h > 0$,*

$$\left|\left|\Phi_{t,h}^{f_1} - \Phi_{t,h}^{f_2}\right|\right|_{L^\infty(\Omega)} \leqslant Ch\,||f_1(t,\cdot) - f_2(t,\cdot)||_{L^\infty(\Omega)} \tag{3.38}$$

for some positive constant $C > 0$ independent from f_1 and f_2.

(ii) *For $f : [0,T]\times\Omega \to \mathbb{R}^d$ smooth enough and $t \in [0,T]$, there exists $L_f > 0$ s.t. for all $y_1, y_2 \in \Omega$, for all $h > 0$, we have*

$$\left|\Phi_{t,h}^{f}(y_1) - \Phi_{t,h}^{f}(y_2)\right| \leqslant (1 + L_f h)|y_1 - y_2| \tag{3.39}$$

Then there exist constants $\alpha_\phi, \lambda, \mu, M, \beta > 0$ (independent from h, ε) s.t. for all $h \leqslant h_+$ and $\varepsilon \in]0,1]$:

$$\begin{aligned} \underset{0\leqslant n\leqslant N}{Max}\left|y^\varepsilon(t_n) - y_{\theta,n}^\varepsilon\right| &\leqslant \delta_\phi\varepsilon + \alpha_\phi\frac{e^{\lambda T}-1}{\lambda}\left[\overline{C}h^{p+q-1} + C\delta_F\right] \\ &+ \frac{e^{\mu T}-1}{\mu}\left[M'h^p + \frac{e^{\lambda T}-1}{\lambda}\beta(\overline{C}h^{p+q-1} + C\delta_F) + C\delta_g\right] \end{aligned} \tag{3.40}$$

Remarks. ***(i)*** *For some nonnegative constants $L_f, \alpha_{\theta,F}$ and α_ϕ independent from h, ε, we get:*

$$\delta_g \leqslant (1 + L_f + \alpha_{\theta,F}\varepsilon)\delta_\phi + \alpha_\phi\delta_F \tag{3.41}$$

(ii) *As flow can be considered for non-autonomous differential equations, it depends on time, but previous estimates concerning numerical flow are assumed to be independent from time (excepted integration time T).*

(iii) *Norm $||\cdot||_{W^{1,\infty}(\Omega),L^\infty([0,2\pi]\times[0,\varepsilon_+])}$ corresponds to $W^{1,\infty}$-Sobolev norm w.r.t. space variable y and L^∞-norm w.r.t. variables τ and ε.*

3.2.2.3 Alternative method

Alternative method for autonomous case yields to specific numerical error bounds.

Theorem 30. *Let consider and denote the following learning errors:*

(i) Learning error for flow ϕ^ε (high oscillations):

$$\delta_\phi := \underset{(\tau,y,\varepsilon)\in[0,2\pi]\times\Omega\times[0,\varepsilon_+]}{Max} \left|\phi_\tau^\varepsilon(y) - \phi_\theta(\tau,y,\varepsilon)\right|. \tag{3.42}$$

(ii) Learning error for flow φ^{g^ε}:

$$\delta_\varphi := \underset{(y,h,\varepsilon)\in\Omega\times[0,h_+]\times[0,\varepsilon_+]}{Max} \left|\frac{\varphi_h^{g^\varepsilon}(y) - \varphi_\theta(y,h,\varepsilon)}{h}\right|. \tag{3.43}$$

Then there exist positive constants λ, L and C_T such that, for all $\varepsilon \in]0,1]$:

$$\underset{0\leqslant n\leqslant N}{Max} \left|y_{\theta,n}^\varepsilon - y^\varepsilon(t_n)\right| \leqslant \delta_\phi + L\frac{e^{\lambda_\theta T}-1}{\lambda}\delta_\varphi + C_T e^{-\frac{C_T}{\varepsilon}} \tag{3.44}$$

3.3 Numerical experiments

To illustrate our theoretical results, we have tested the method given in subsection 3.2.1 for a simple dynamical system taken from physics:

1. **Inverted Pendulum:** This dynamical system describes the evolution of an unstable pendulum with the center of gravity above its pivot point, experiencing forced oscillations. It is governed by the following equation:

$$\left\{\begin{array}{rcl} \dot{y_1^\varepsilon}(t) &=& y_2^\varepsilon(t) + \sin\left(\frac{t}{\varepsilon}\right)\sin\left(y_1^\varepsilon(t)\right) \\ \dot{y_2^\varepsilon}(t) &=& \sin\left(y_1^\varepsilon(t)\right) - \frac{1}{2}\sin\left(\frac{t}{\varepsilon}\right)^2\sin\left(2y_1^\varepsilon(t)\right) - \sin\left(\frac{t}{\varepsilon}\right)\cos\left(y_1^\varepsilon(t)\right)y_2^\varepsilon(t) \end{array}\right. \tag{3.45}$$

   and the average field associated to this equation is given by

$$\langle f\rangle(y) = \begin{bmatrix} y_2 \\ \sin(y_1) - \frac{1}{4}\sin(2y_1) \end{bmatrix} \tag{3.46}$$

2. **Van der Pol oscillator:** This is a system that describes an electrical system with nonlinear damping. It is governed by the following two-dimensional system:

$$\left\{\begin{array}{rcl} \dot{q} & = & \frac{1}{\varepsilon}p \\ \dot{p} & = & -\frac{1}{\varepsilon}q + \left(\frac{1}{4} - q^2\right)p \end{array}\right. \tag{3.47}$$

by making the variable change $(y_1, y_2) \longmapsto S\left(\frac{t}{\varepsilon}\right)(q,p)$, where $\tau \longmapsto S(\tau)$ is given by

$$\tau \longmapsto S(\tau) = \begin{bmatrix} \cos(\tau) & -\sin(\tau) \\ \sin(\tau) & \cos(\tau) \end{bmatrix} \tag{3.48}$$

we get the system:

$$\left\{\begin{array}{rcl} \dot{y_1}(t) & = & -\sin\left(\frac{t}{\varepsilon}\right)\left[\frac{1}{4} - \left(y_1(t)\cos\left(\frac{t}{\varepsilon}\right) + y_2(t)\sin\left(\frac{t}{\varepsilon}\right)\right)^2\right]\left[-y_1(t)\sin\left(\frac{t}{\varepsilon}\right) + y_2(t)\cos\left(\frac{t}{\varepsilon}\right)\right] \\ \dot{y_2}(t) & = & \cos\left(\frac{t}{\varepsilon}\right)\left[\frac{1}{4} - \left(y_1(t)\cos\left(\frac{t}{\varepsilon}\right) + y_2(t)\sin\left(\frac{t}{\varepsilon}\right)\right)^2\right]\left[-y_1(t)\sin\left(\frac{t}{\varepsilon}\right) + y_2(t)\cos\left(\frac{t}{\varepsilon}\right)\right] \end{array}\right. \tag{3.49}$$

which is the classical form for highly oscillatory systems. The average field of the system in its canonical form is given, for all $y \in \mathbb{R}^2$, by

$$\langle f\rangle(y) = \frac{1}{8}(1 - |y|^2)y. \tag{3.50}$$

For a highly oscillatory differential equation of the form (3.1), one can approximate the averaged field F^ε and the highly oscillatory generator ϕ^ε (defined by formal power series (3.3)) by using the following approximation sequences [21]:

$$\left\{\begin{array}{rcl} \phi_\tau^{[0]}(y) & = & y \\ \phi_\tau^{[k+1]}(y) & = & y + \varepsilon\displaystyle\int_0^\tau f\left(\sigma, \phi_\sigma^{[k]}(y)\right) - \frac{\partial \phi_\sigma^{[k]}}{\partial y}(y)F^{[k]}(y)\mathrm{d}\sigma \end{array}\right. \tag{3.51}$$

and

$$\left\{\begin{array}{rcl} F^{[0]}(y) & = & \langle f\rangle(y) \\ F^{[k]}(y) & = & \left(\frac{\partial\langle\phi_\cdot^{[k]}\rangle}{\partial y}(y)\right)^{-1}\left\langle f\left(\cdot, \phi_\cdot^{[k]}(y)\right)\right\rangle. \end{array}\right. \tag{3.52}$$

These sequences can be used to obtain an arbitrary order approximation of formal power series (3.3)

$$\phi_\tau^\varepsilon(y) = \phi_\tau^{[k]}(y) + \mathcal{O}\left(\varepsilon^{k+1}\right) \text{ and } F^\varepsilon(y) = F^{[k]}(y) + \mathcal{O}\left(\varepsilon^{k+1}\right) \tag{3.53}$$

3.3.1 Approximation of the averaged field and high oscillation generator

In this subsection, we investigate the approximation error between the learned averaged field and the high oscillation generator with the theoretical averaged field and the high oscillation generator, both at orders 0 and 1 for the inverted pendulum. We examine the learning error with respect to the high oscillation parameter variables ε. Specifically, we compute the values

$$\underset{y\in\Omega}{Max}\left|F_\theta\left(y,0,\varepsilon\right)-F^{[k]}(y)\right| \text{ and } \underset{(\tau,y)\in[0,2\pi]\times\Omega}{Max}\left|\phi_\theta\left(\tau,y,\varepsilon\right)-\phi_\tau^{[k]}(y)\right| \tag{3.54}$$

where $\phi^{[k]}$ and $F^{[k]}$ are computed by the formulas (3.51) and (3.52) for $k=0,1$ respectively.

Figures 3.1, 3.2 and 3.3 confirm that the modified field can be adequately learned with our neural network. However, the learning error seems to slow down the decay of the error bounds.

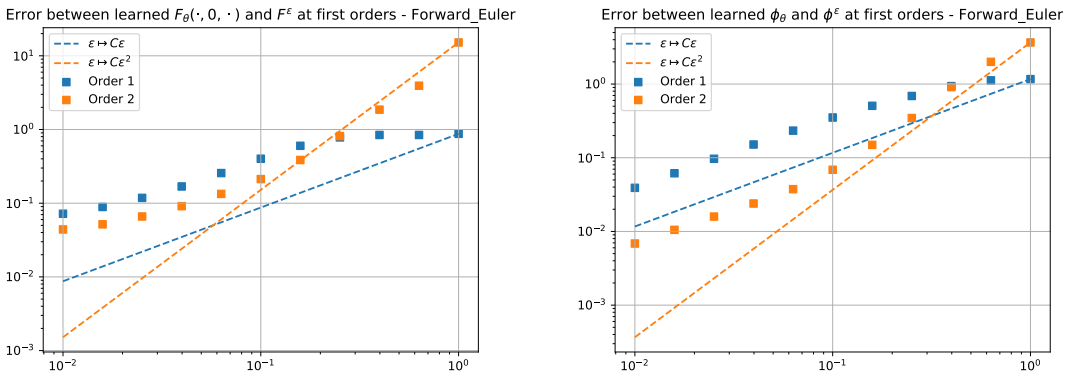

Figure 3.1 – Inverted pendulum with forward Euler method. Left: Error between $F^{[k]}$ and $F_\theta(\cdot,0,\varepsilon)$ for $k=0,1$ w.r.t. ε. Right: Error between $\phi^{[k]}$ and $\phi_\theta(\cdot,\cdot,\varepsilon)$ for $k=0,1$ w.r.t. ε.

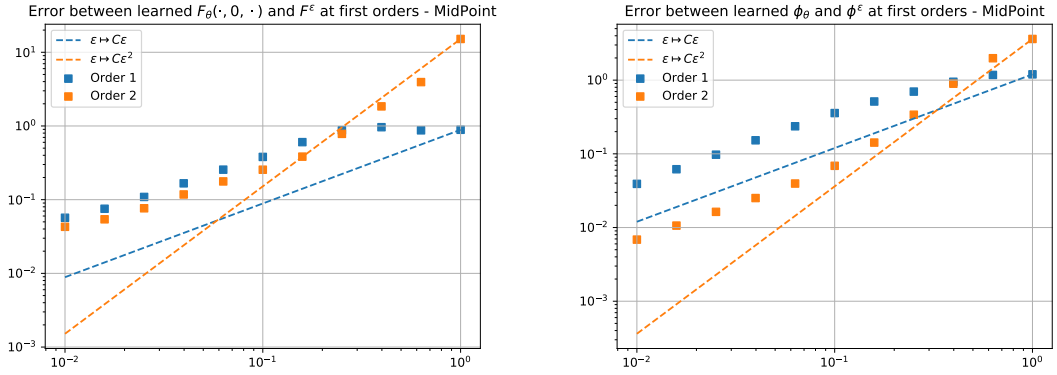

Figure 3.2 – Inverted pendulum with midpoint method. Left: Error between $F^{[k]}$ and $F_\theta(\cdot, 0, \varepsilon)$ for $k = 0, 1$ w.r.t. ε. Right: Error between $\phi^{[k]}$ and $\phi_\theta(\cdot, \cdot, \varepsilon)$ for $k = 0, 1$ w.r.t. ε.

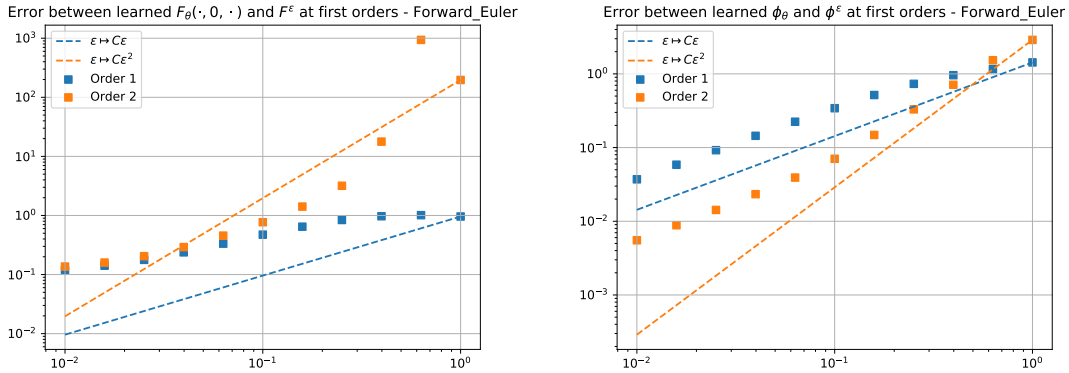

Figure 3.3 – Van der Pol oscillator with forward Euler method. Left: Error between $F^{[k]}$ and $F_\theta(\cdot, 0, \varepsilon)$ for $k = 0, 1$ w.r.t. ε. Right: Error between $\phi^{[k]}$ and $\phi_\theta(\cdot, \cdot, \varepsilon)$ for $k = 0, 1$ w.r.t. ε.

3.3.2 Loss decay and integration of ODE's

In order to compare the integration of a dynamical system with the learned modified averaged field and the learned high oscillation generator, we will now solve the inverted pendulum with the Forward Euler and Midpoint methods. Before doing so, we study the decay of the loss functions for the training and test data sets ($Loss_{Train}$ and $Loss_{Test}$). Their similarity is a good indication that there is no overfitting (the

size of the training data set is thus appropriately estimated). As the Mean Squared Error (MSE) loss is used, it gives an idea of the value of the square of the error for the equation and the auto-encoder structure.

Figures 3.4, 3.5, 3.8, and 3.9 show an accurate numerical integration with the slow-fast decomposition-based method by using the corresponding learned vector field for both the Forward Euler and Midpoint methods for the inverted pendulum. For the Van der Pol oscillator, if the learning for the transformed dynamical system appears to be less accurate (figures 3.12 and 3.14), the integration after the inverse variable change (3.48) seems to be correct, as shown in figures 3.13 and 3.15.

However, adding the micro-macro correction greatly increases the accuracy, especially for larger values of ε. In particular, the numerical approximation of the solution of the inverted pendulum is more accurate, as we can observe in figures 3.6, 3.7, 3.10, and 3.11. For the Van der Pol oscillator, it can be observed that the solution of the system is accurately approximated by using the micro-macro correction, as shown in figures 3.16 and 3.18, even before the variable change, as can be seen in figures 3.17 and 3.19.

3.3.2.1 Inverted Pendulum - forward Euler method

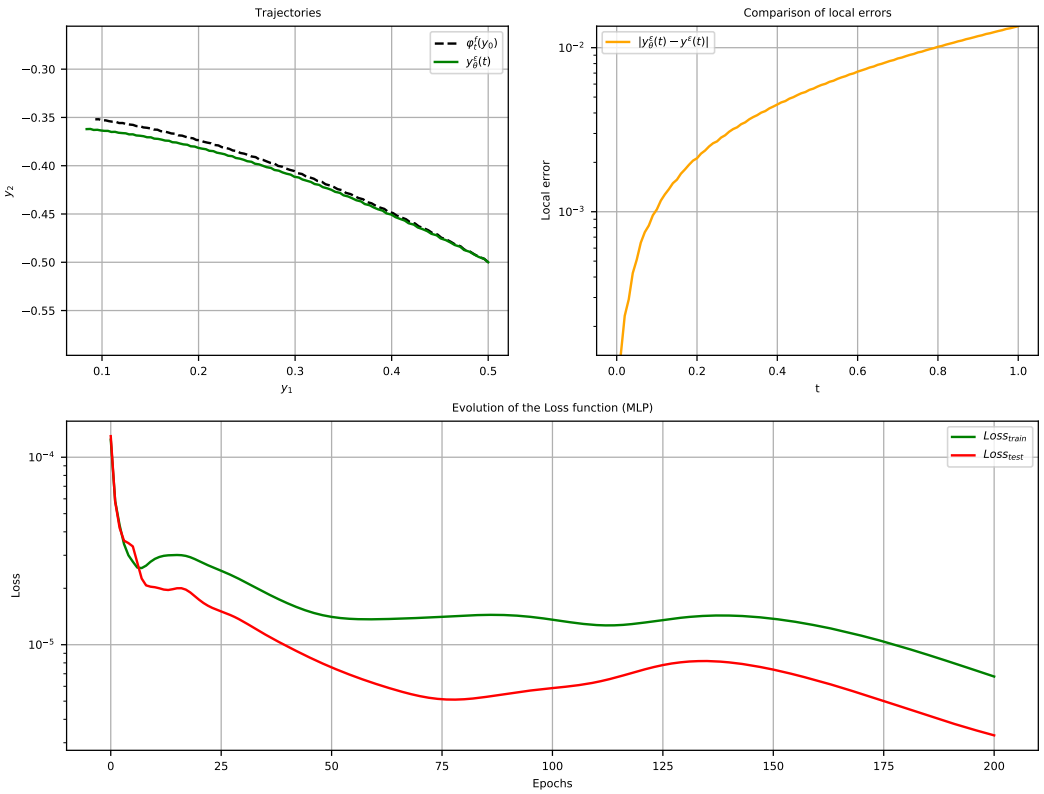

Figure 3.4 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the inverted pendulum with Forward Euler method in the case $\varepsilon = 0.001$.

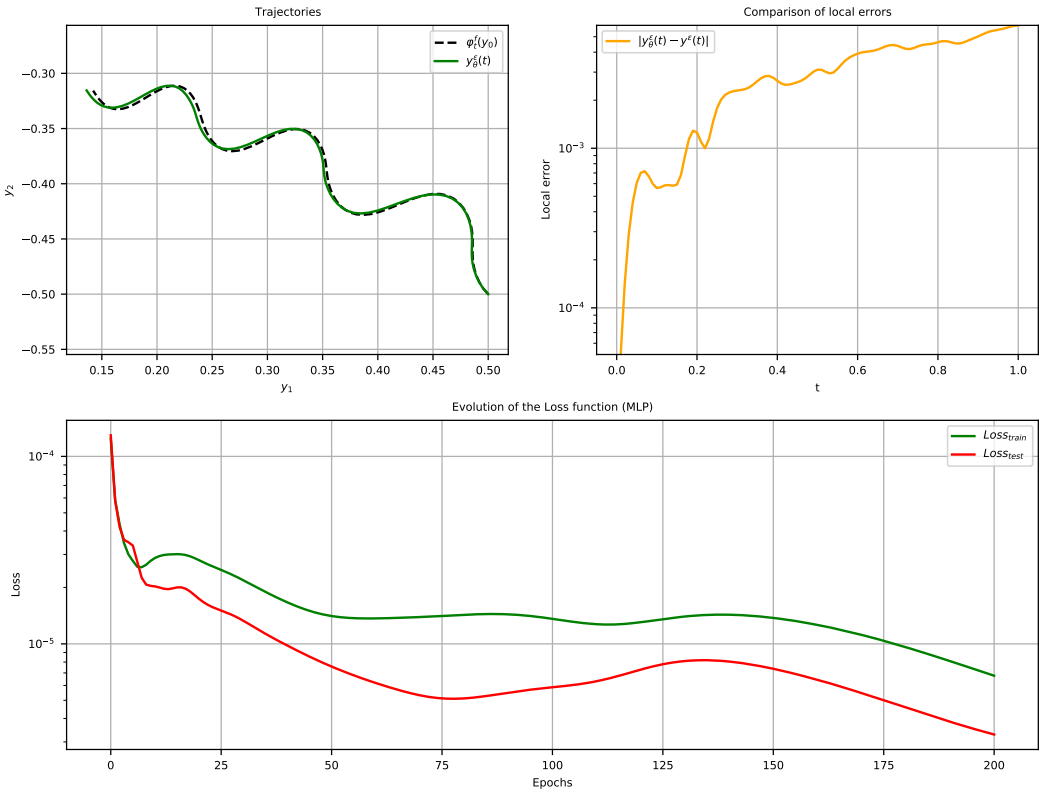

Figure 3.5 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the inverted pendulum with Forward Euler method in the case $\varepsilon = 0.05$.

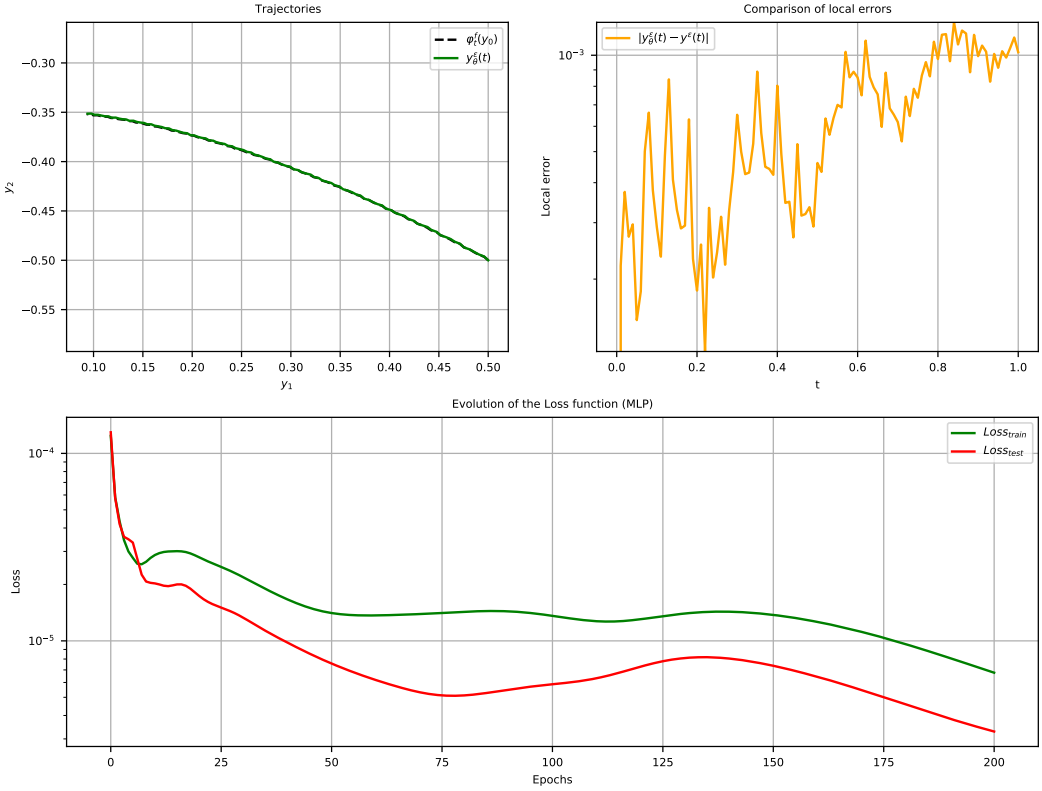

Figure 3.6 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the inverted pendulum with Forward Euler method with Micro-Macro correction in the case $\varepsilon = 0.001$.

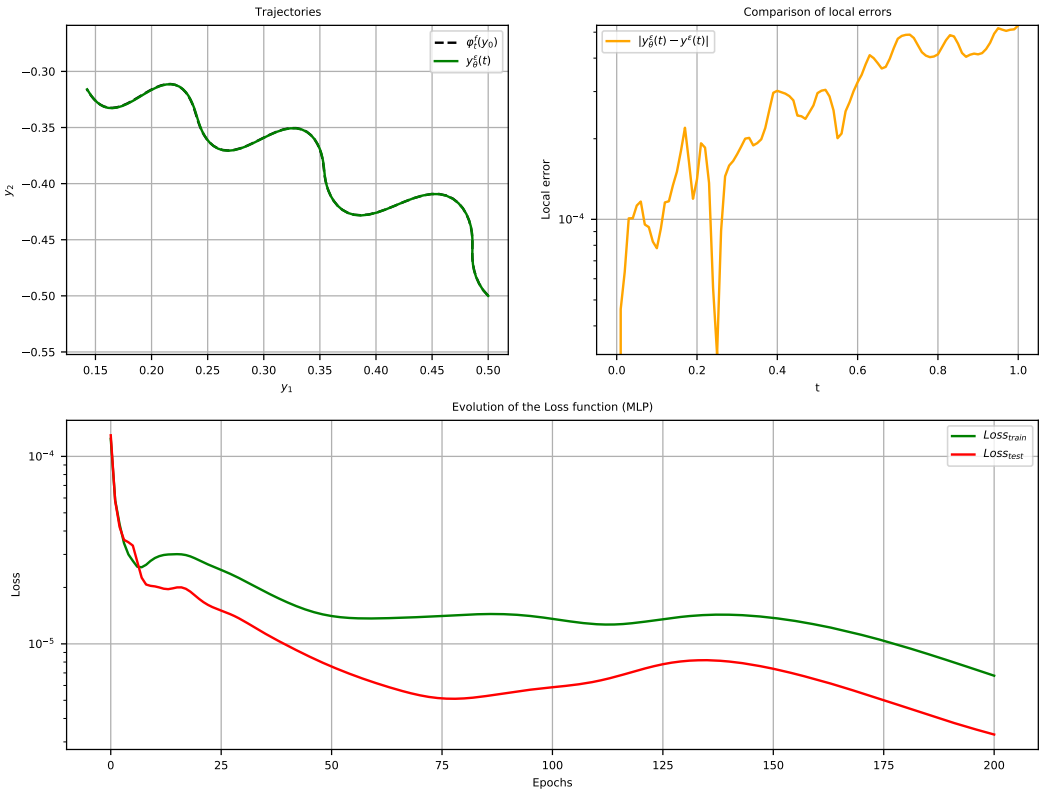

Figure 3.7 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the inverted pendulum with Forward Euler method with Micro-Macro correction in the case $\varepsilon = 0.05$.

3.3.2.2 Inverted Pendulum - midpoint method

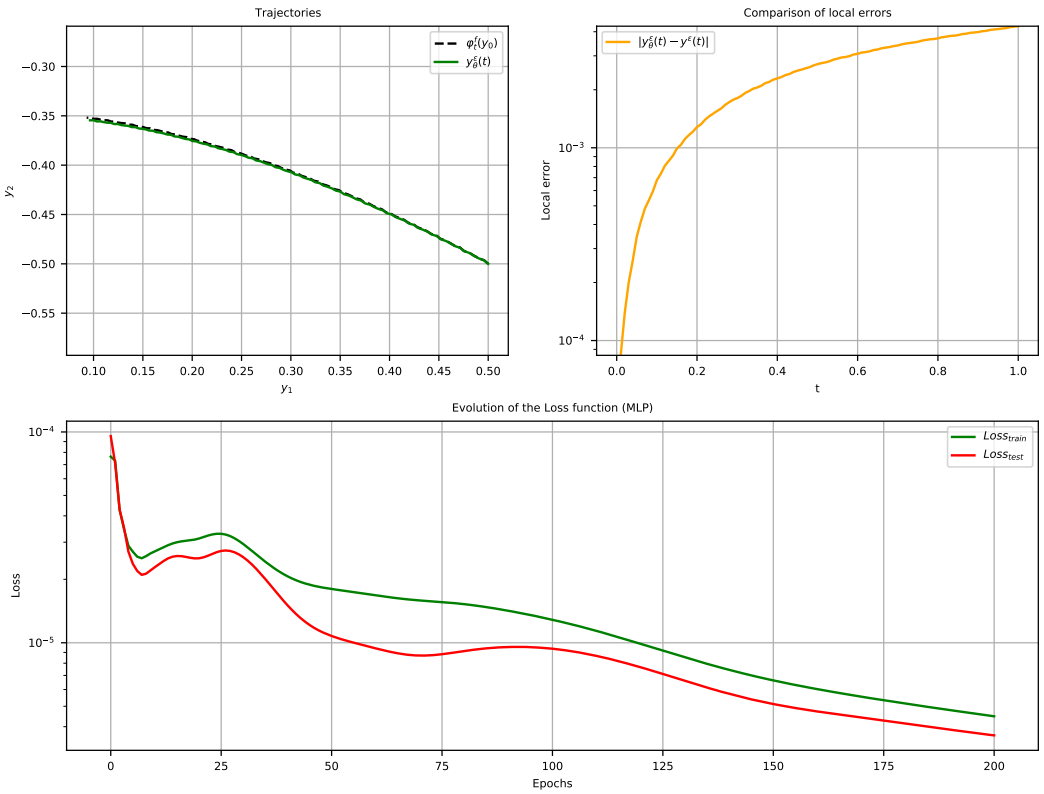

Figure 3.8 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the inverted pendulum with midpoint method in the case $\varepsilon = 0.001$.

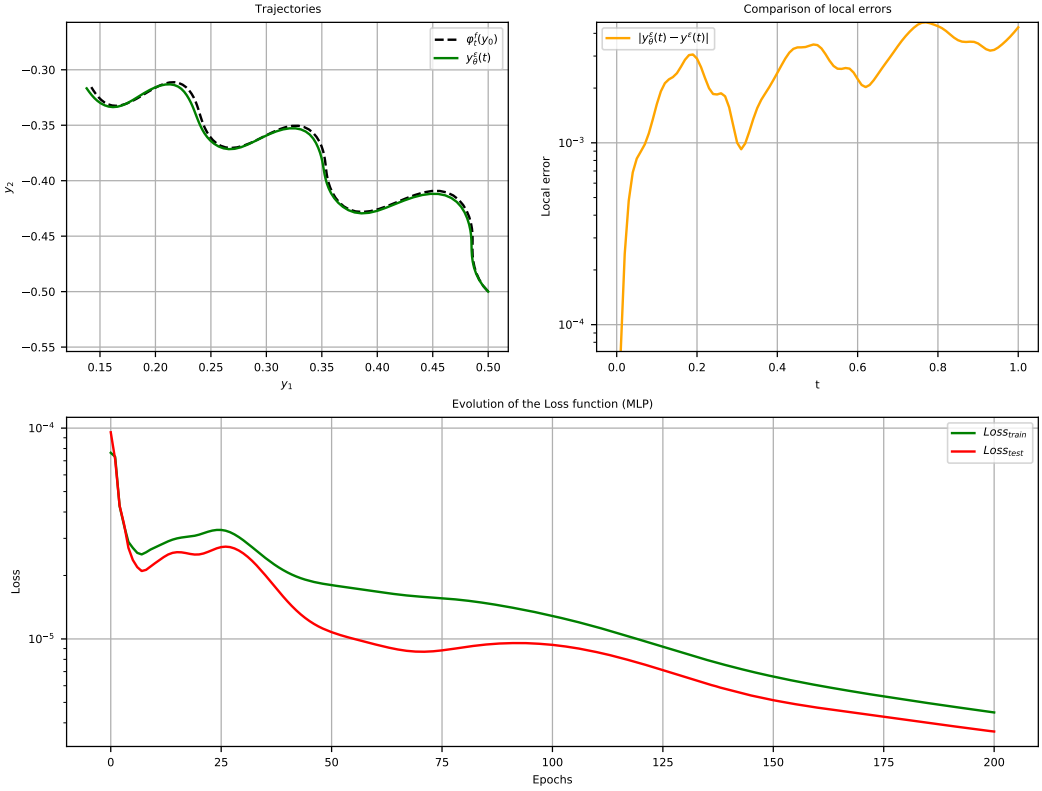

Figure 3.9 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the inverted pendulum with midpoint method in the case $\varepsilon = 0.05$.

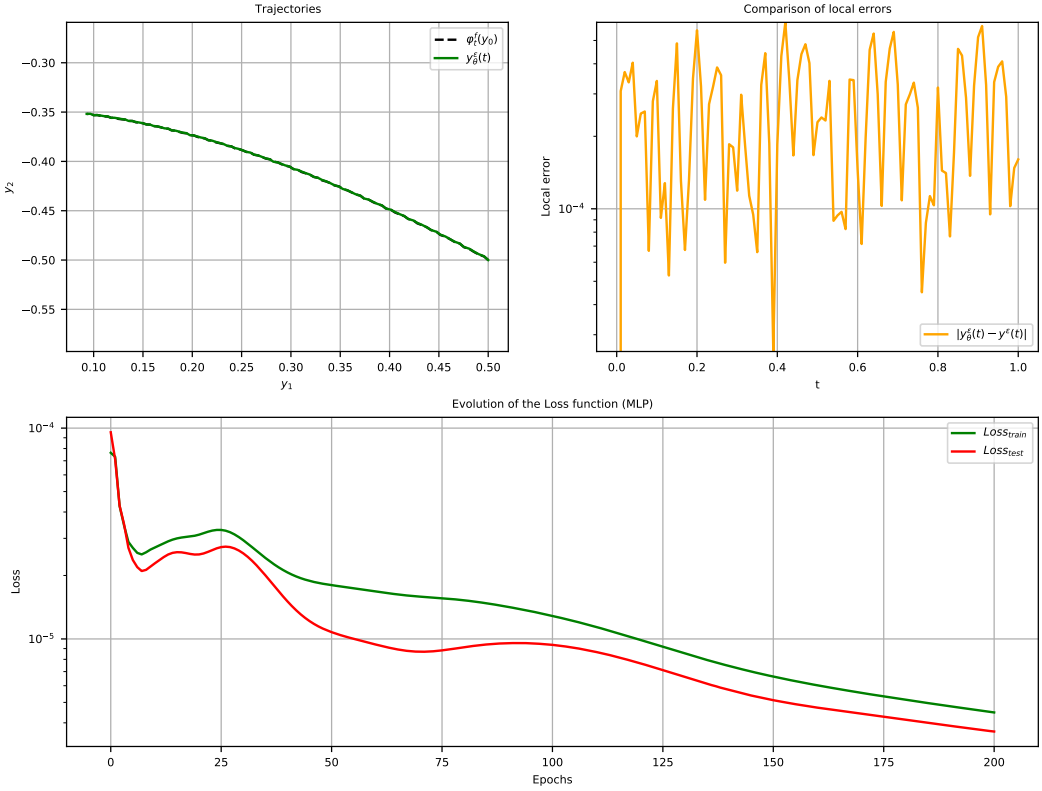

Figure 3.10 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the inverted pendulum with midpoint method with Micro-Macro correction in the case $\varepsilon = 0.001$.

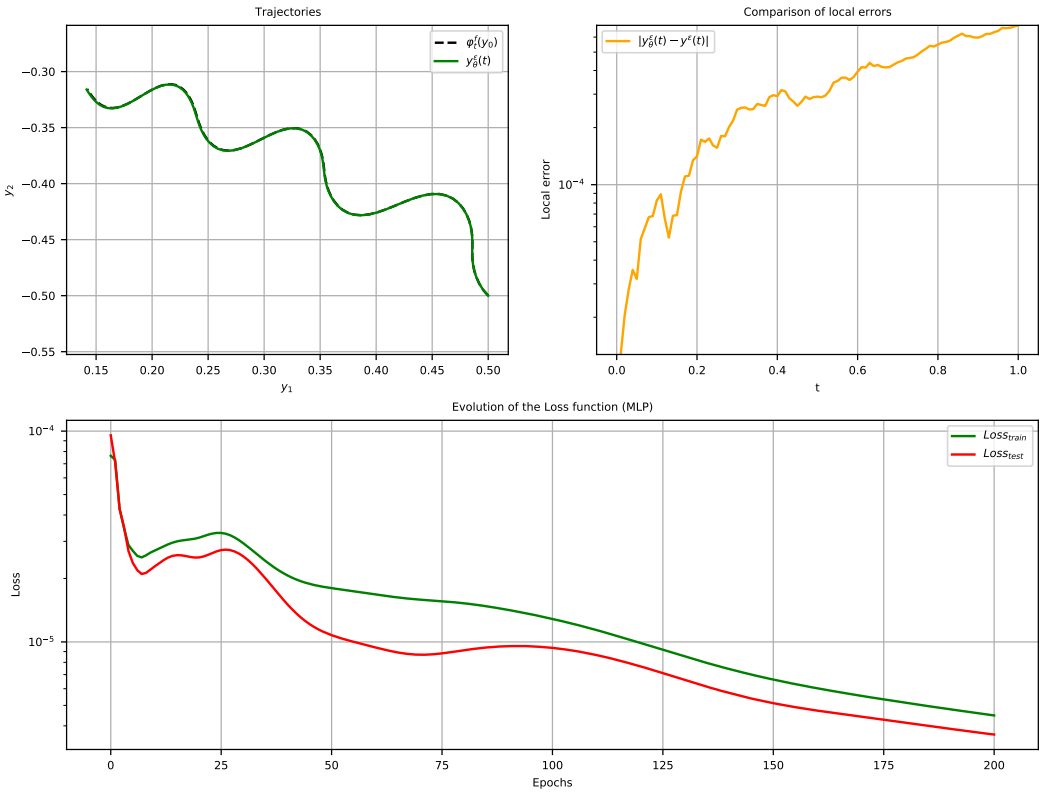

Figure 3.11 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the inverted pendulum with midpoint method with Micro-Macro correction in the case $\varepsilon = 0.05$.

3.3.2.3 Van der Pol oscillator - forward Euler method

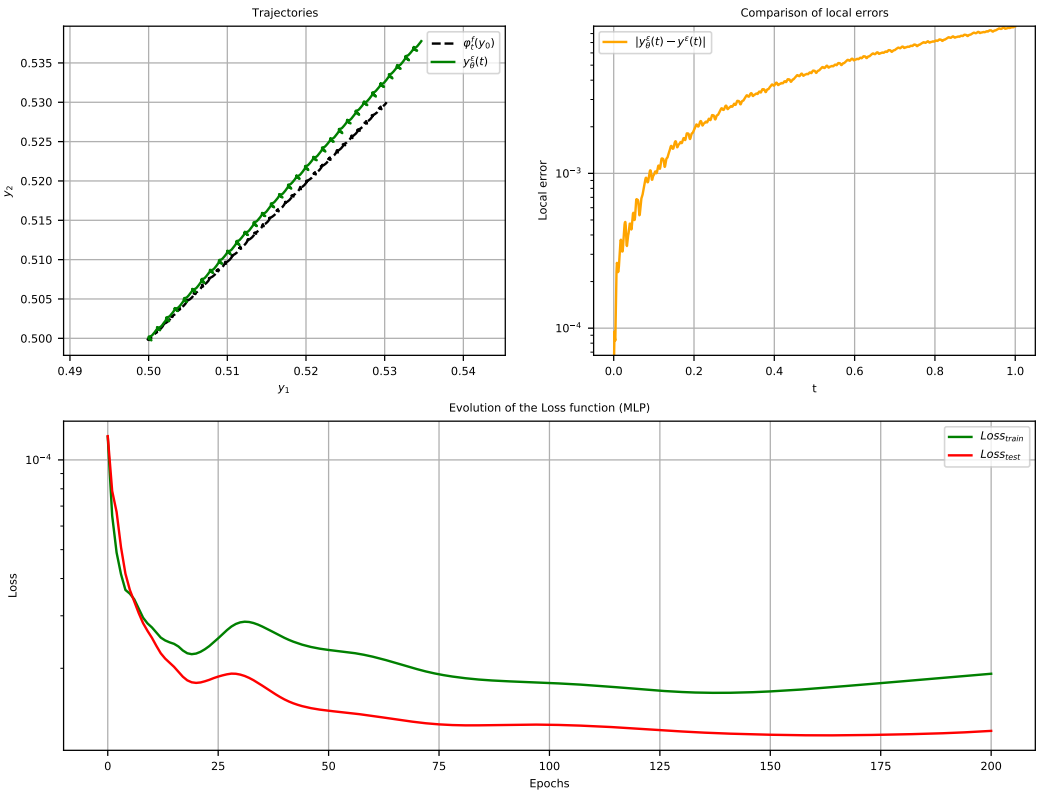

Figure 3.12 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the Van der Pol oscillator with Forward Euler method in the case $\varepsilon = 0.01$.

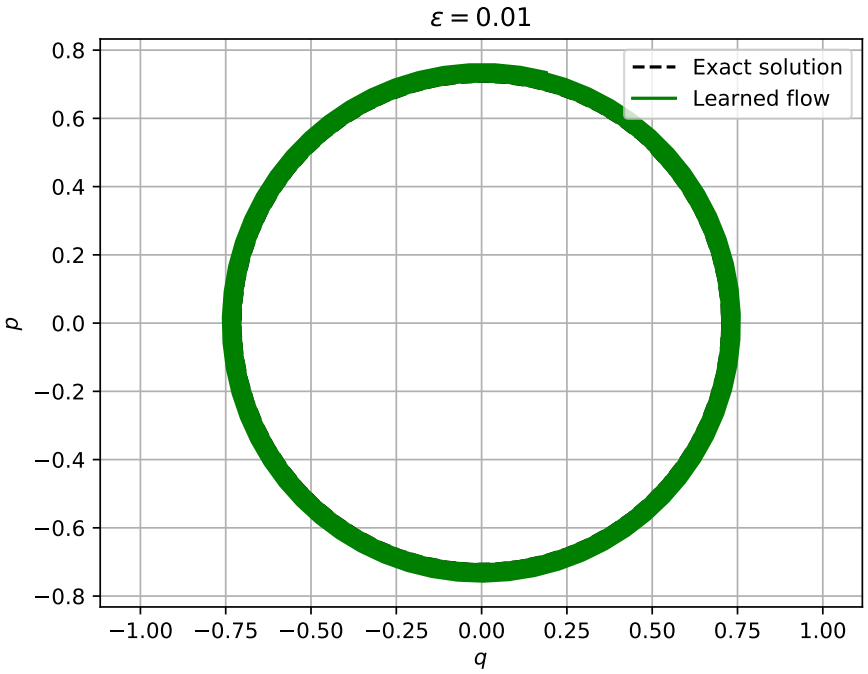

Figure 3.13 – Comparison between trajectories (dashed dark: exact flow, green: numerical flow with learned vector field) for the Van der Pol oscillator with Forward Euler method in the case $\varepsilon = 0.01$ after the inverse variable change (3.48).

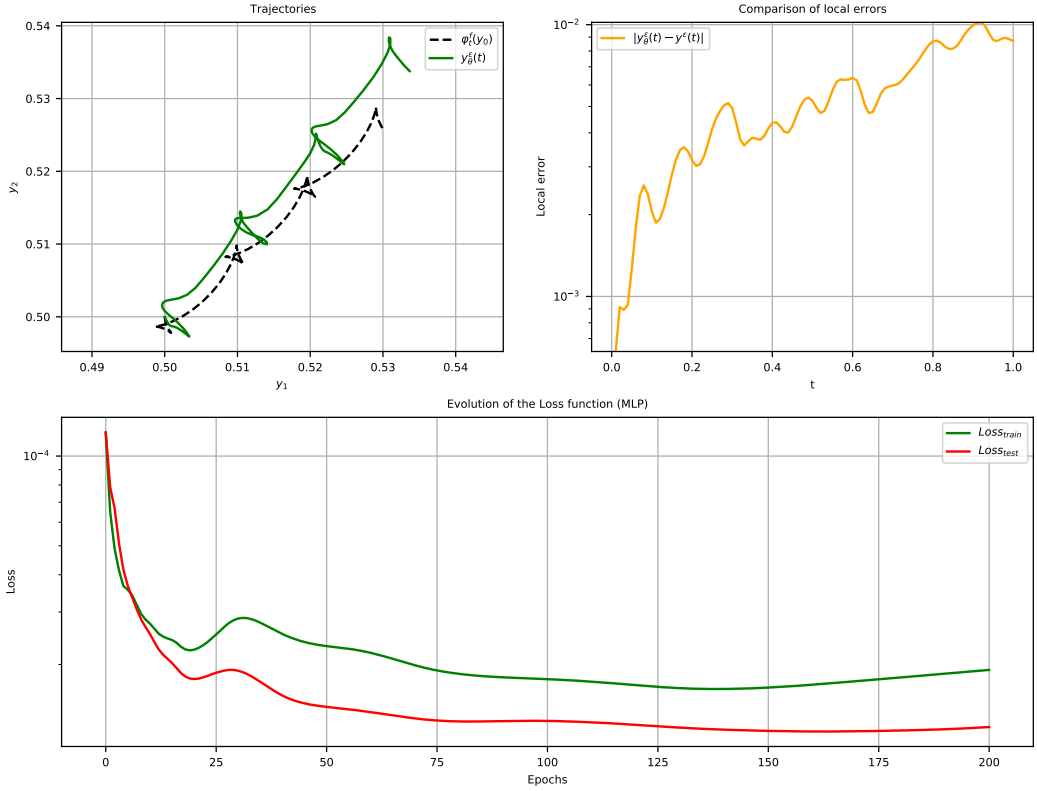

Figure 3.14 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the Van der Pol oscillator with Forward Euler method in the case $\varepsilon = 0.1$.

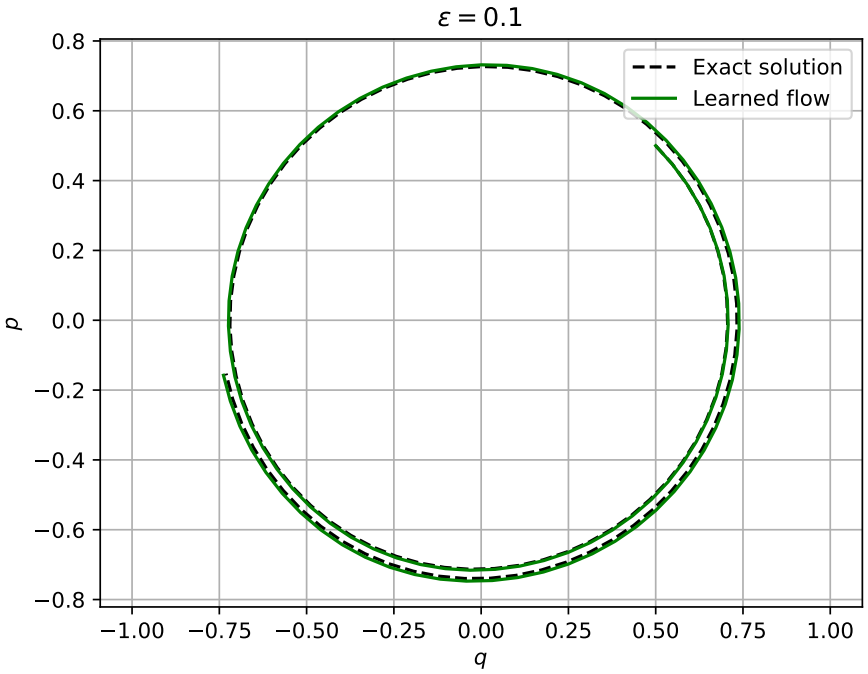

Figure 3.15 – Comparison between trajectories (dashed dark: exact flow, green: numerical flow with learned vector field) for the Van der Pol oscillator with Forward Euler method in the case $\varepsilon = 0.1$ after the inverse variable change (3.48).

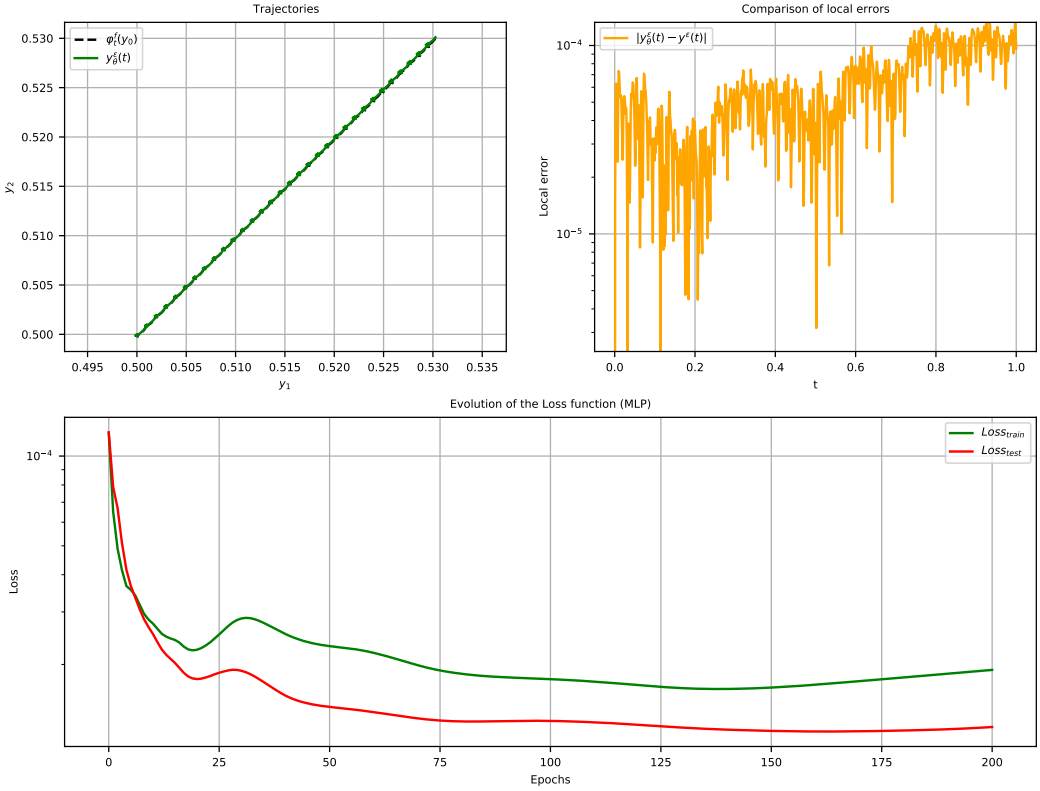

Figure 3.16 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the Van der Pol oscillator with Forward Euler method with Micro-Macro correction in the case $\varepsilon = 0.01$.

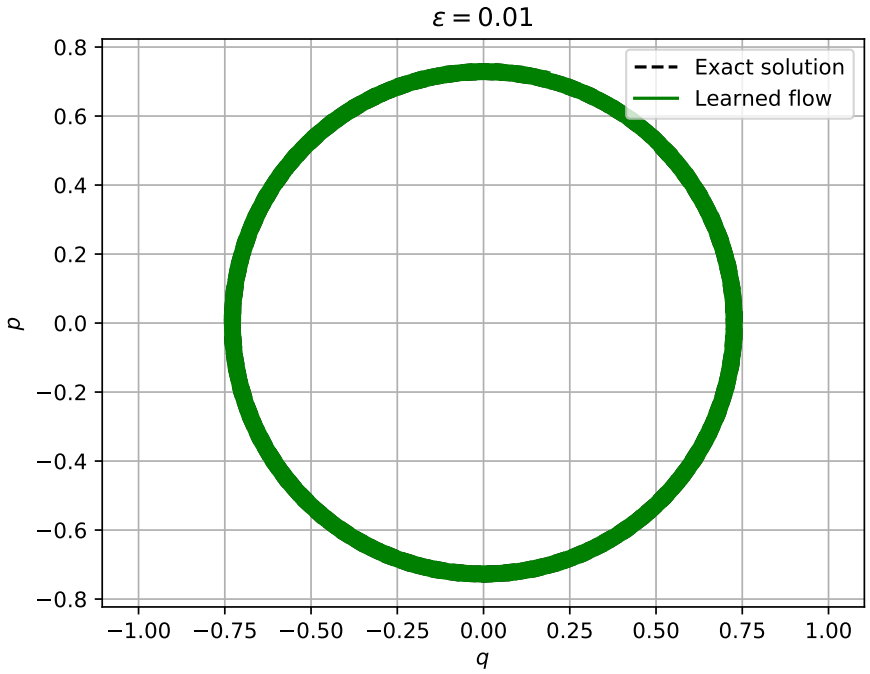

Figure 3.17 – Comparison between trajectories (dashed dark: exact flow, green: numerical flow with learned vector field) for the Van der Pol oscillator with Forward Euler method with Micro-Macro correction in the case $\varepsilon = 0.01$ after the inverse variable change (3.48).

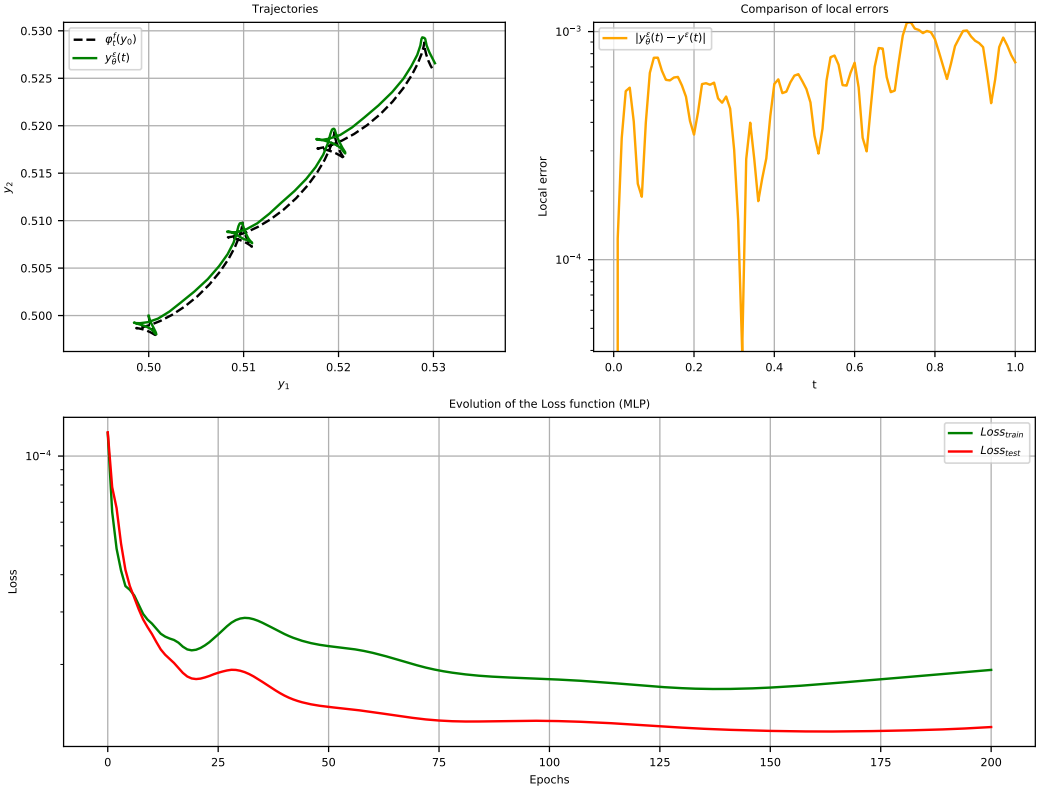

Figure 3.18 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the Van der Pol oscillator with Forward Euler method with Micro-Macro correction in the case $\varepsilon = 0.1$.

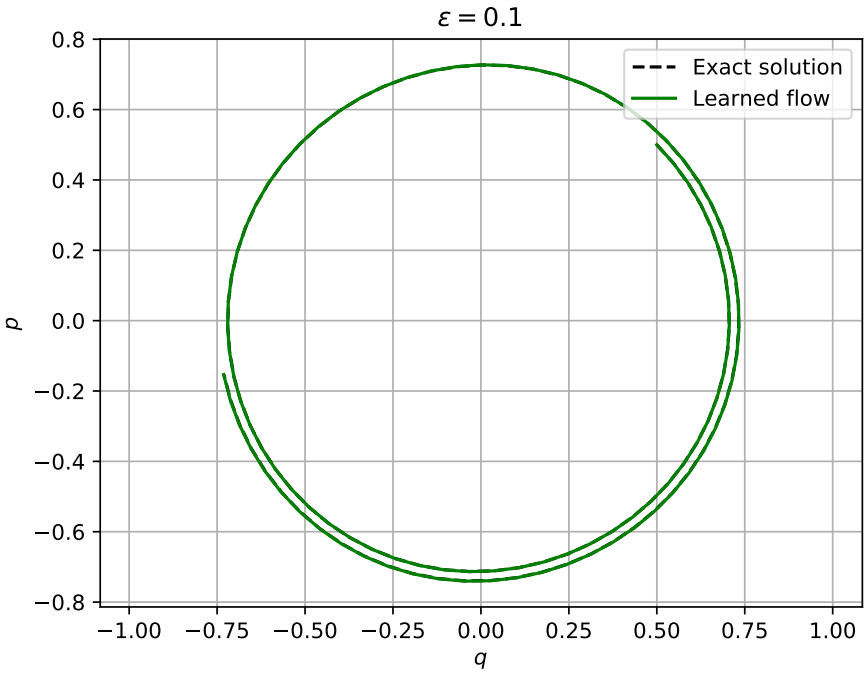

Figure 3.19 – Comparison between trajectories (dashed dark: exact flow, green: numerical flow with learned vector field) for the Van der Pol oscillator with Forward Euler method with Micro-Macro correction in the case $\varepsilon = 0.1$ after the inverse variable change (3.48).

3.3.3 Error curves w.r.t. step size

We also study the global error between the exact flow and the numerical flow obtained from the learned vector fields. The errors are plotted as a function of the step size, and the curves are in perfect agreement with the estimates of the previous theorems (for the Forward Euler and Midpoint methods). Figures 3.20, 3.21, and 3.22 show that numerical integration using micro-macro correction is more accurate than the *slow-fast* decomposition-based method and confirm the results obtained in the previous subsection 3.3.2.

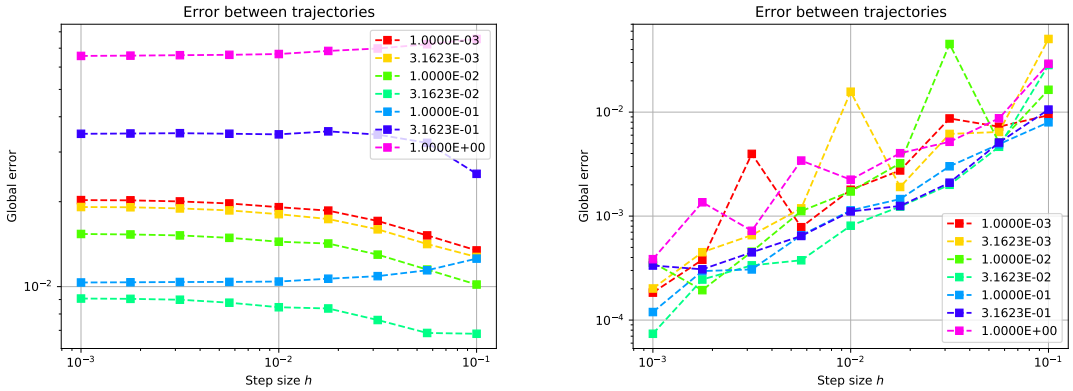

Figure 3.20 – Integration errors (each color corresponds to a high oscillation parameter ε) of Inverted Pendulum with Forward Euler. Left: *Slow-fast* decomposition-based method. Right: With Micro-Macro correction.

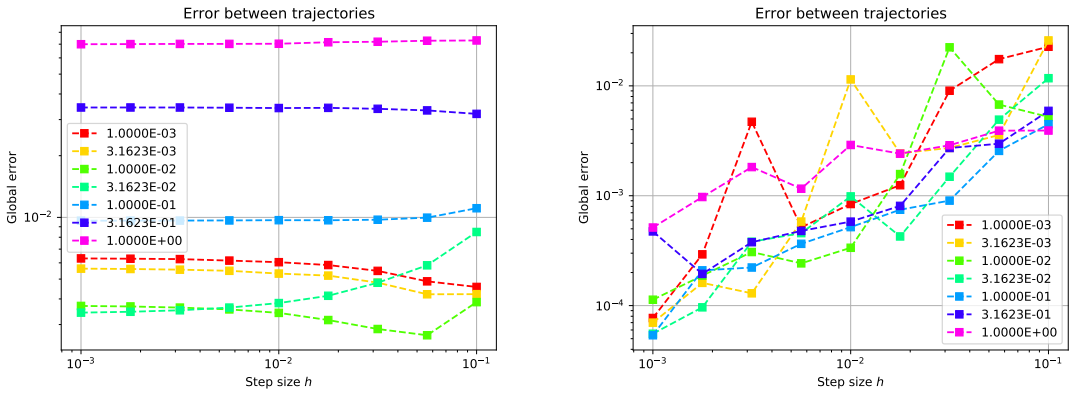

Figure 3.21 – Integration errors (each color corresponds to a high oscillation parameter ε) of Inverted Pendulum with midpoint. Left: *Slow-fast* decomposition-based method. Right: With Micro-Macro correction.

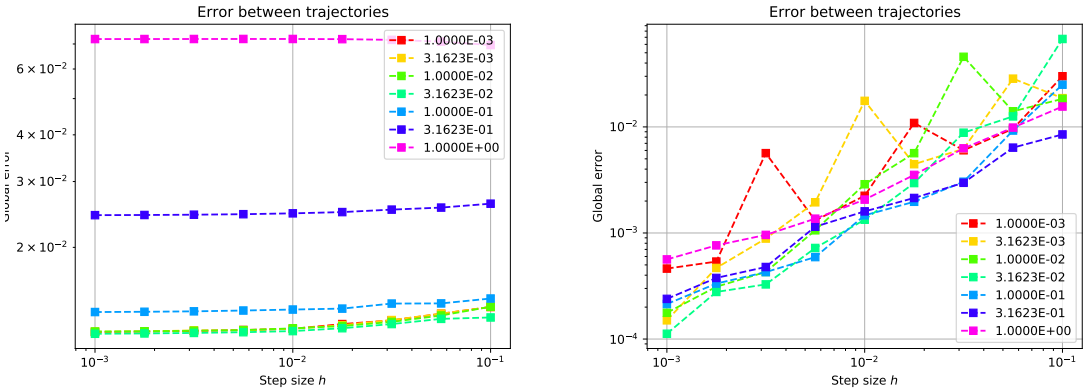

Figure 3.22 – Integration errors (each color corresponds to a high oscillation parameter ε) of Van der Pol oscillator with Forward Euler. Left: *Slow-fast* decomposition-based method. Right: With Micro-Macro correction.

3.3.4 Uniform accuracy test

As the micro-macro method is presented as a *uniformly accurate* (UA) method [21, 22], one can check whether the micro-macro correction with machine learning has uniform accuracy by plotting the global errors vs. the parameter ε for several step sizes h.

Figures 3.23, 3.24, and 3.25 show that uniform accuracy is almost verified with micro-macro correction. However, we don't have this property with the *slow-fast* decomposition-based method, especially for larger values of ε. This phenomenon is due to the exponential remainder in the formula (3.5).

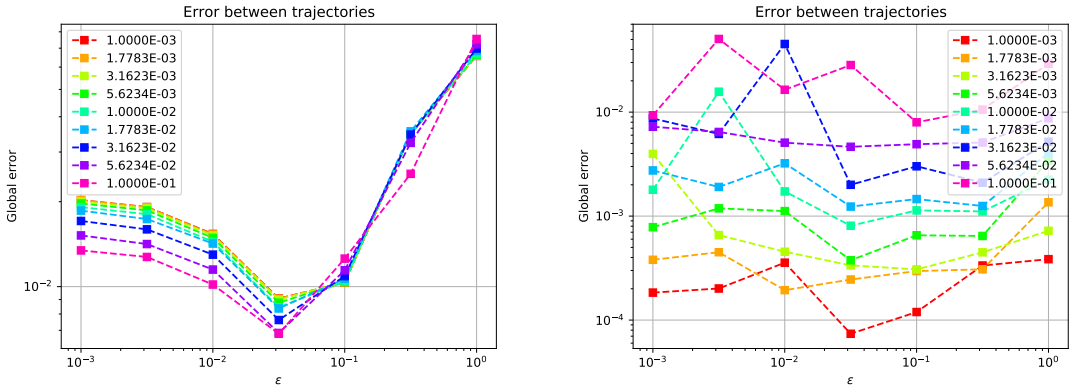

Figure 3.23 – Uniform accuracy test (each color corresponds to a step size h) of Inverted Pendulum system with Forward Euler. Left: *Slow-fast* decomposition-based method. Right: With Micro-Macro correction.

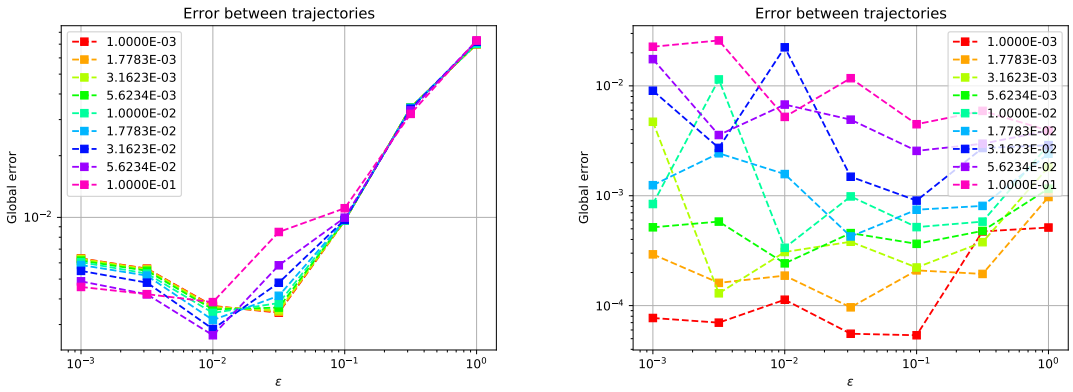

Figure 3.24 – Uniform accuracy test (each color corresponds to a step size h) of Inverted Pendulum system with midpoint. Left: *Slow-fast* decomposition-based method. Right: With Micro-Macro correction.

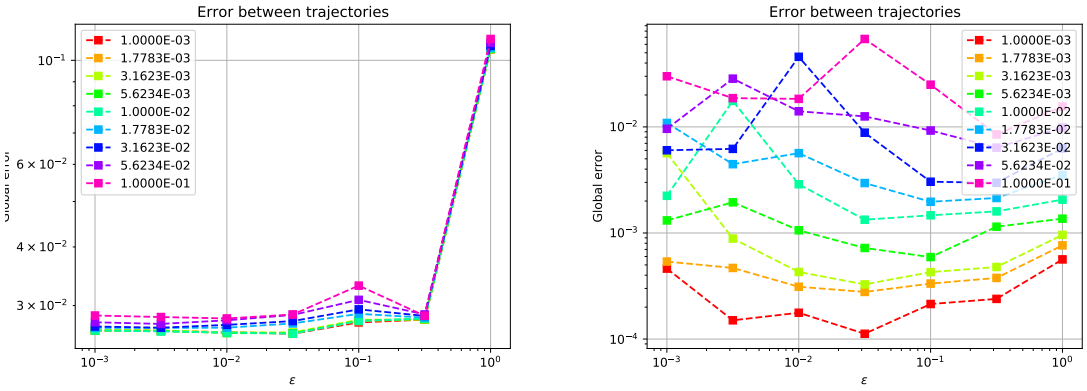

Figure 3.25 – Uniform accuracy test (each color corresponds to a step size h) of Van der Pol oscillator with Forward Euler. Left: *Slow-fast* decomposition-based method. Right: With Micro-Macro correction.

3.3.5 Evaluation of alternative method

In this subsection, we make a comparison between the classical method using *slow-fast* decomposition and the alternative method in the autonomous case, following the example of the Van der Pol oscillator. Despite the absence of an auto-encoder, the learning of the solution seems to be more accurate for the classical method (Figures 3.26, 3.27, 3.28, and 3.29) than the alternative method (Figures 3.30 and 3.31). Furthermore, Figure 3.32 confirms the difference in accuracy.

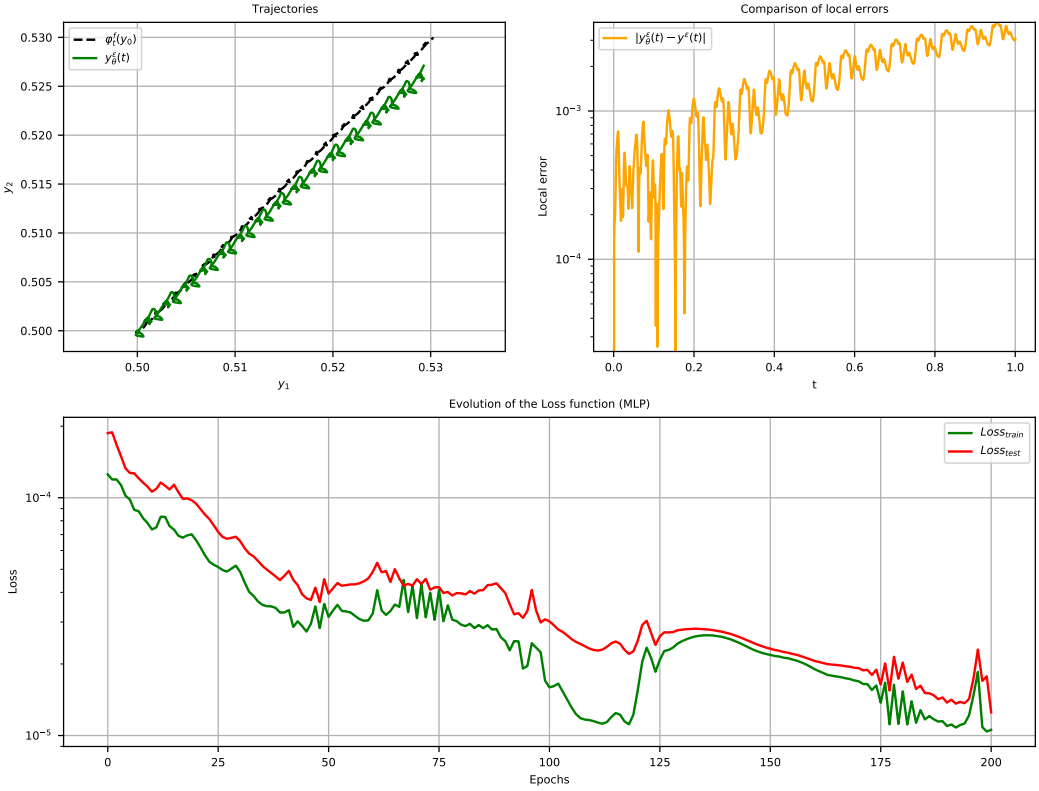

Figure 3.26 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the Van der Pol oscillator with Forward Euler method in the case $\varepsilon = 0.01$.

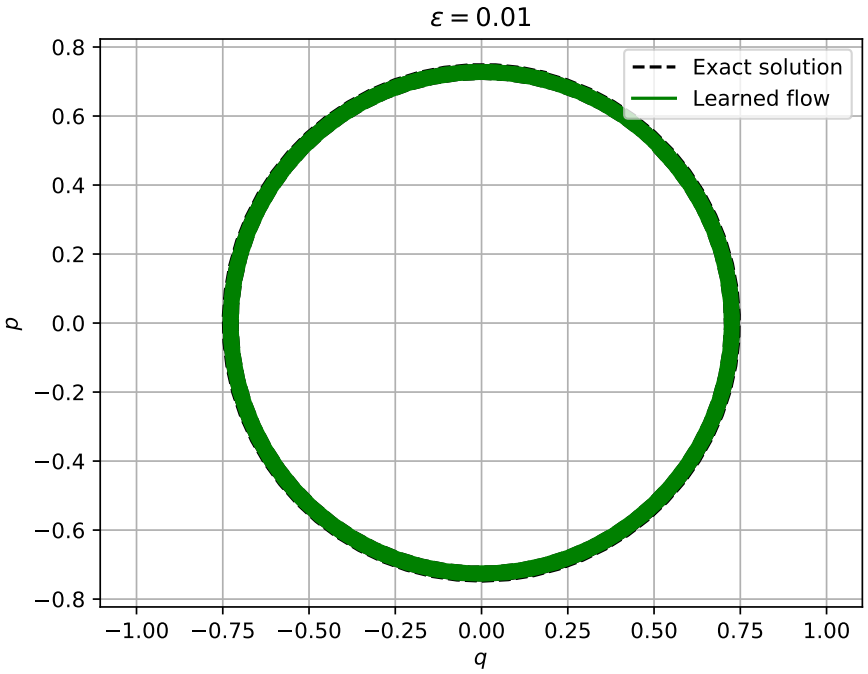

Figure 3.27 – Comparison between trajectories (dashed dark: exact flow, green: numerical flow with learned vector field) for the Van der Pol oscillator with Forward Euler method in the case $\varepsilon = 0.01$ after the inverse variable change (3.48).

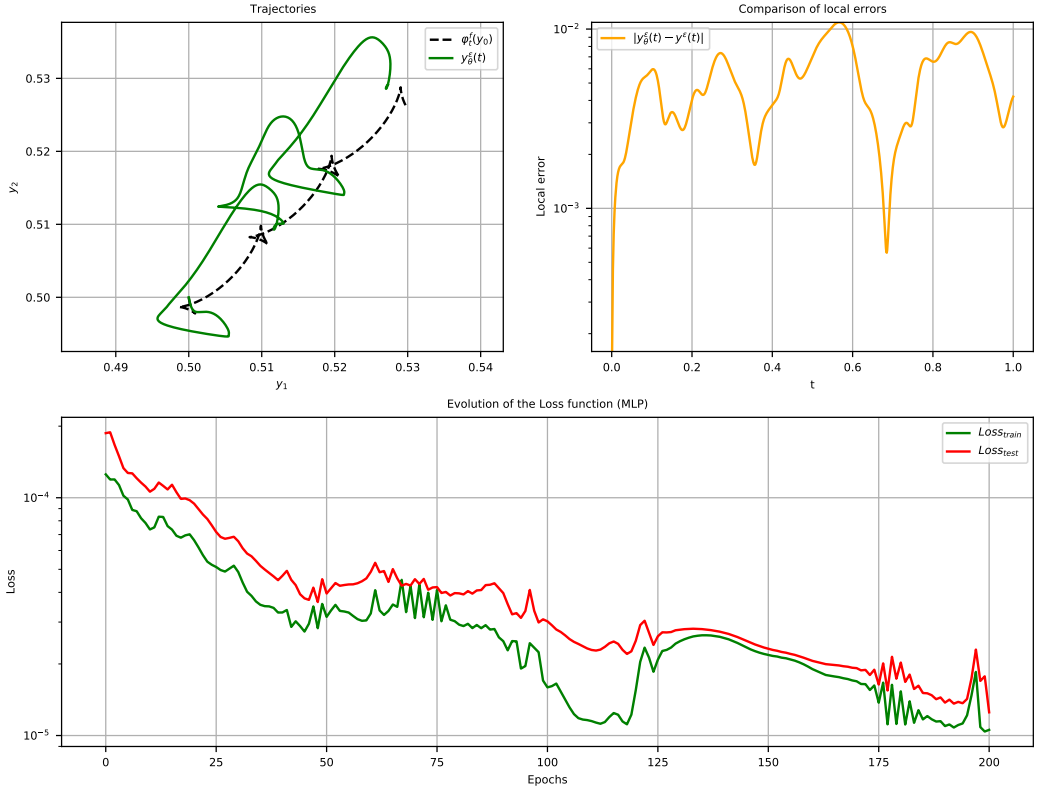

Figure 3.28 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the Van der Pol oscillator with Forward Euler method in the case $\varepsilon = 0.1$.

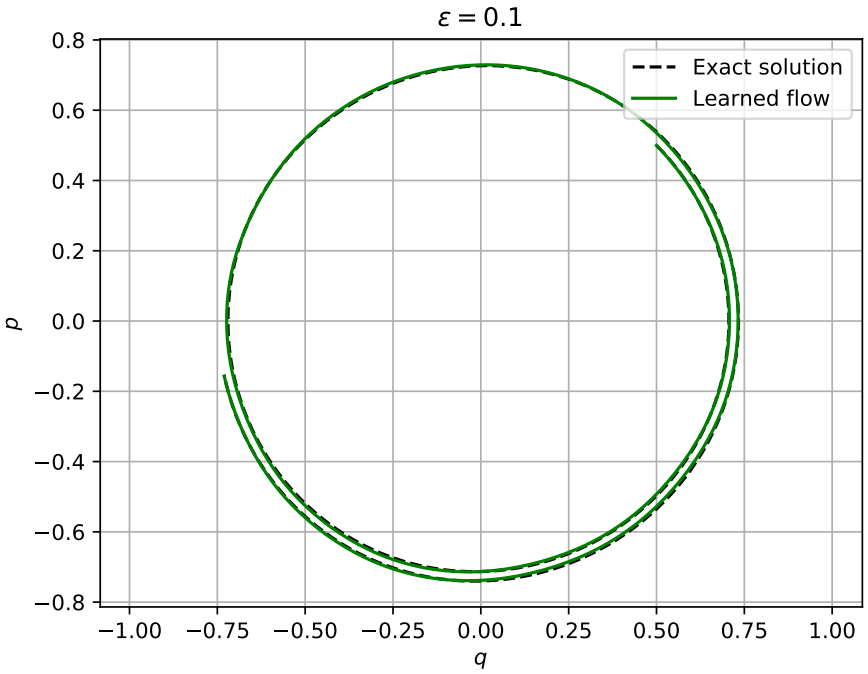

Figure 3.29 – Comparison between trajectories (dashed dark: exact flow, green: numerical flow with learned vector field) for the Van der Pol oscillator with Forward Euler method in the case $\varepsilon = 0.1$ after the inverse variable change (3.48).

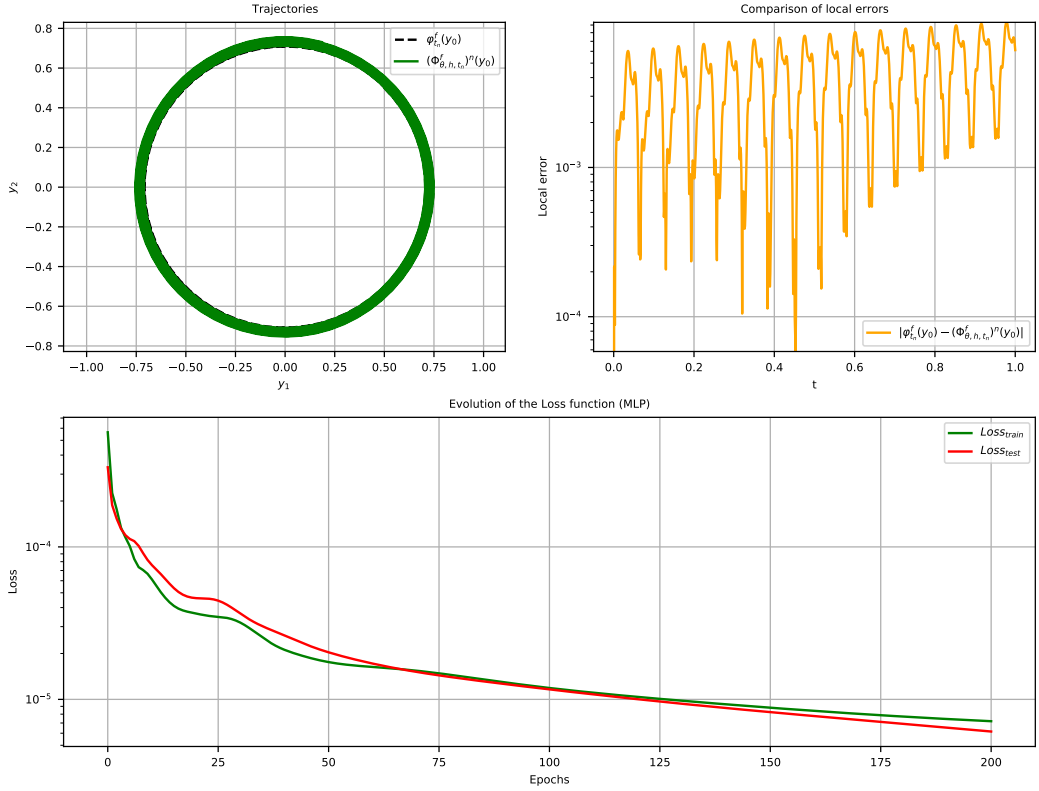

Figure 3.30 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the Van der Pol oscillator with alternative method in the case $\varepsilon = 0.01$.

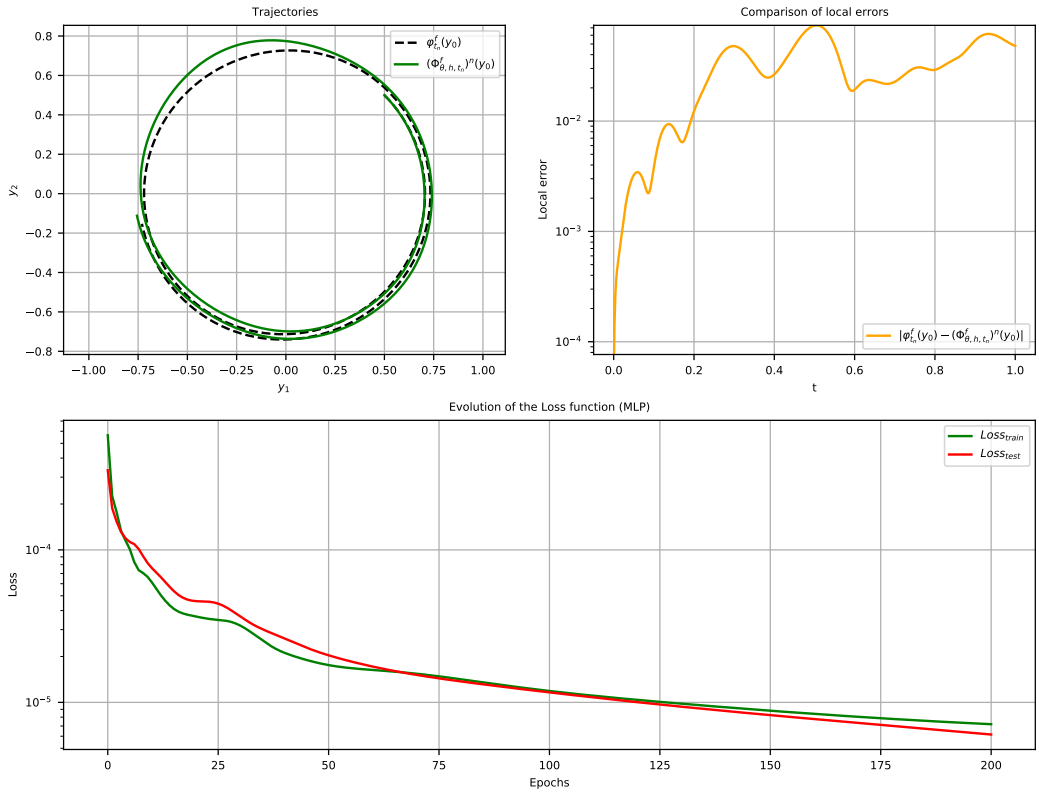

Figure 3.31 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, green: numerical flow with learned vector fields and local error (yellow) for the Van der Pol oscillator with alternative method in the case $\varepsilon = 0.1$.

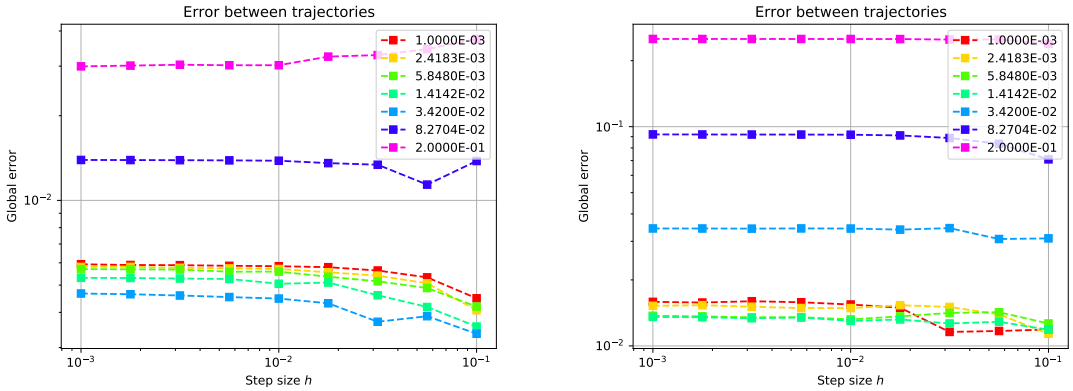

Figure 3.32 – Integration errors (each color corresponds to a high oscillation parameter ε) of Van der Pol oscillator. Left: *Slow-fast* decomposition-based method. Right: Alternative method for autonomous case.

3.4 Conclusions

The numerical experiments presented in this paper demonstrate the feasibility of learning the highly oscillatory generator and the averaged field using neural networks. Furthermore, numerical integration without pre-computation leads to accurate approximations of exact solutions of highly oscillatory differential equations. Although the method based on *slow-fast* decomposition is simpler and requires fewer computations than micro-macro correction, it exhibits nearly uniform accuracy and consistency (although perfect uniform accuracy and consistency are hindered by learning errors). Given that high-dimensional problems pose greater challenges for machine learning, our methods are primarily suitable for small-dimensional systems.

3.A Appendix: Proof of Theorem 28

Let $t \mapsto \psi_t^\varepsilon$ be the exact solution of the autonomous differential equation associated with the vector field $F^{\varepsilon,[n_\varepsilon]}$. Furthermore, let $\psi_{\theta,n}^\varepsilon$ denote the following numerical flow:

$$\left\{ \begin{array}{rcl} \psi_{\theta,0}^\varepsilon & = & y^\varepsilon(0) \\ \psi_{\theta,n+1}^\varepsilon & = & \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(\psi_{\theta,n}^\varepsilon) \end{array} \right. \tag{3.55}$$

First, we study the estimates over this autonomous ODE. Then a general error estimate is given.

(i). Consistency error (autonomous ODE): Consistency error is given by

$$\begin{aligned} \varepsilon_{\theta,n} &:= \psi^{\varepsilon}_{t_{n+1}} - \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(\psi^{\varepsilon}_{t_n}) \\ &= \varphi_h^{F^{\varepsilon,[n_\varepsilon]}}(\psi^{\varepsilon}_{t_n}) - \Phi_h^{\widetilde{F_h^{\varepsilon,[n_\varepsilon],[q]}}}(\psi^{\varepsilon}_{t_n}) + \Phi_h^{\widetilde{F_h^{\varepsilon,[n_\varepsilon],[q]}}}(\psi^{\varepsilon}_{t_n}) - \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(\psi^{\varepsilon}_{t_n}) \end{aligned} \tag{3.56}$$

According to the first hypothesis of theorem 28, we get

$$|\varepsilon_{\theta,n}| \leqslant \overline{C} h^{p+q} + C\delta_F h \tag{3.57}$$

(ii). Local truncation error (autonomous ODE): Local truncation error is given by

$$e_{\theta,n} := \psi^{\varepsilon}_{\theta,n} - \psi^{\varepsilon}_{t_n}. \tag{3.58}$$

Thus we have

$$e_{\theta,n+1} = \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(\psi^{\varepsilon}_{\theta,n}) - \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(\psi^{\varepsilon}_{t_n}) - \varepsilon_{\theta,n}. \tag{3.59}$$

Thus there exists a constant λ (Lipschitz constant of F_θ with respect to its space variable) such that

$$|e_{\theta,n+1}| \leqslant (1+\lambda h)|e_{\theta,n}| + \overline{C} h^{p+q} + C\delta_F h. \tag{3.60}$$

Using the dicrete Grönwall lemma we get

$$\underset{0\leqslant n\leqslant N}{Max}|e_{\theta,n}| \leqslant \frac{e^{\lambda T}-1}{\lambda}\left[\overline{C} h^{p+q-1} + C\delta_F\right] \tag{3.61}$$

(iii). General error estimate: Let consider and denote $\eta_{\theta,n}$ the following error

$$\begin{aligned} \eta_{\theta,n} &:= y^{\varepsilon}(t_n) - y^{\varepsilon}_{\theta,n} \\ &= y^{\varepsilon}(t_n) - \phi_\theta\left(\frac{t_n}{\varepsilon}, \psi^{\varepsilon}_{\theta,n}, \varepsilon\right). \end{aligned} \tag{3.62}$$

By using the following decomposition:

$$\begin{aligned} \eta_{\theta,n} &= y^{\varepsilon}(t_n) - \phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_n}{\varepsilon}}(\psi^{\varepsilon}_{t_n}) + \phi^{\varepsilon,[n_\varepsilon]}_{\frac{t_n}{\varepsilon}}(\psi^{\varepsilon}_{t_n}) - \phi_\theta\left(\frac{t_n}{\varepsilon}, \psi^{\varepsilon}_{t_n}, \varepsilon\right) \\ &+ \phi_\theta\left(\frac{t_n}{\varepsilon}, \psi^{\varepsilon}_{t_n}, \varepsilon\right) - \phi_\theta\left(\frac{t_n}{\varepsilon}, \psi^{\varepsilon}_{\theta,n}, \varepsilon\right), \end{aligned} \tag{3.63}$$

we bound the first, second and third terms from above using (3.5), the learning error δ_ϕ and the estimate over $e_{\theta,n}$. This will give us the estimate we want.

3.B Appendix: Proof of Theorem 29

1. **Error estimate for $v_{\theta,n}$:**

   For the estimate of v, we make a classical error estimation used for autonomous differential equations, similar to the estimate of the autonomous part done in the proof of theorem 28.

   (i). Consistency Error: Consistency error is given by

$$\begin{aligned} \varepsilon_{\theta,v,n} &:= v(t_{n+1}) - \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(v(t_n)) \\ &= \varphi_h^{F^{[p]}}(v(t_n)) - \Phi_h^{\widetilde{F^{[p],[q]}}}(v(t_n)) + \Phi_h^{\widetilde{F^{[p],[q]}}}(v(t_n)) - \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(v(t_n)) \end{aligned} \tag{3.64}$$

   according to the first hypothesis of Theorem 29 we get

$$|\varepsilon_{\theta,v,n}| \leqslant \overline{C} h^{p+q} + C\delta_F h. \tag{3.65}$$

   (ii). Local Truncation Error: Local truncation error is given by

$$e_{\theta,v,n} \;:=\; v(t_n) - v_{\theta,n}. \tag{3.66}$$

Thus we have

$$\begin{aligned} e_{\theta,v,n+1} &:= v(t_{n+1}) - v_{\theta,n+1} \\ &= \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(v(t_n)) - \Phi_h^{F_\theta(\cdot,h,\varepsilon)}(v_{\theta,n}) + \varepsilon_{\theta,v,n}. \end{aligned} \tag{3.67}$$

Thus, there exists a constant λ (Lipschitz constant of F_θ w.r.t. space variable) such that:

$$|e_{\theta,v,n+1}| \;\leqslant\; (1+\lambda h)|e_{v,\theta,n}| + \overline{C}h^{p+q} + C\delta_F h. \tag{3.68}$$

Using the discrete Grönwall lemma, we have

$$\underset{0\leqslant n\leqslant N}{Max}|e_{\theta,v,n}| \;\leqslant\; \frac{e^{\lambda T}-1}{\lambda}\left[\overline{C}h^{p+q-1} + C\delta_F\right] \tag{3.69}$$

2. Error estimate for $w_{\theta,n}$:

For the estimate of w, we treat v as a source term and perform an error estimation similar to the case of autonomous differential equations.

(i). Consistency Error: Consistency error is given by

$$\begin{aligned} \varepsilon_{\theta,w,n} &:= w(t_{n+1}) - \Phi_{t_n,h}^{g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v(t_n)\right)}(w(t_n)) \\ &= \underbrace{w(t_{n+1}) - \Phi_{t_n,h}^{g\left(\frac{t_n}{\varepsilon},\cdot,v(t_n)\right)}(w(t_n))}_{\text{Consistency error for classical scheme}} \\ &+ \Phi_{t_n,h}^{g\left(\frac{t_n}{\varepsilon},\cdot,v(t_n)\right)}(w(t_n)) - \Phi_{t_n,h}^{g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v(t_n)\right)}(w(t_n)). \end{aligned} \tag{3.70}$$

Since the first term can be bounded from above by $M'h^{p+1}$ (where M' is a constant independent of h, ε), and the second term can be bounded

from above by using the first hypothesis of theorem 29, we get

$$|e_{\theta,w,n}| \leqslant M'h^{p+1} + C\delta_g h \tag{3.71}$$

(ii). Local Truncation Error: Local truncation error is given by

$$e_{\theta,w,n} = w(t_n) - w_{\theta,n}. \tag{3.72}$$

Thus we have

$$\begin{aligned} e_{\theta,w,n+1} &= w(t_{n+1}) - w_{\theta,n+1} \\ &= \Phi_{t_n,h}^{g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v(t_n)\right)}(w(t_n)) - \Phi_{t_n,h}^{g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v_{\theta,n}\right)}(w_{\theta,n}) + \varepsilon_{\theta,w,n} \\ &= \Phi_{t_n,h}^{g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v(t_n)\right)}(w(t_n)) - \Phi_{t_n,h}^{g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v_{\theta,n}\right)}(w(t_n)) \\ &+ \Phi_{t_n,h}^{g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v_{\theta,n}\right)}(w(t_n)) - \Phi_{t_n,h}^{g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v_{\theta,n}\right)}(w_{\theta,n}) \\ &+ \varepsilon_{\theta,w,n} \end{aligned} \tag{3.73}$$

The first term can be bounded from above using the first hypothesis of theorem 29, the second term can be estimated by introducing μ, the Lipschitz constant of g_θ, and using the second hypothesis of theorem 29. Thus we get

$$\begin{aligned} |e_{\theta,w,n+1}| &\leqslant Ch\left\|g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v(t_n)\right) - g_\theta\left(\frac{t_n}{\varepsilon},\cdot,v_{\theta,n}\right)\right\|_{L^\infty(\Omega)} \\ &+ (1+\mu h)|e_{\theta,w,n}| + M'h^{p+1} + C\delta_g h \\ &\leqslant C\beta h|e_{\theta,v,n}| + (1+\mu h)|e_{\theta,w,n}| + M'h^{p+1} + C\delta_g h, \end{aligned} \tag{3.74}$$

where β is the Lipschitz constant of g_θ with respect to v. Since $|e_{\theta,w,n}|$ can be bounded from above by using the error estimate for v, we get

$$\begin{aligned} |e_{\theta,w,n+1}| &\leqslant (1+\mu h)|e_{\theta,w,n}| + M'h^{p+1} \\ &+ h\left[\frac{e^{\lambda T}-1}{\lambda}\beta(\overline{C}h^{p+q-1}+C\delta_F)+C\delta_g\right]. \end{aligned} \tag{3.75}$$

Using the discrete Grönwall lemma, we get

$$\underset{0\leqslant n\leqslant N}{Max}|e_{\theta,w,n}| \leqslant \frac{e^{\mu T}-1}{\mu}\left[M'h^p+\frac{e^{\lambda T}-1}{\lambda}\beta(\overline{C}h^{p+q-1}+C\delta_F)+C\delta_g\right] \tag{3.76}$$

3. **Error estimate for $y^{\varepsilon}_{\theta,n}$:** Finally, we make an error estimate over y using the formula (3.16) to describe our numerical method. Since we have

$$y^{\varepsilon}_{\theta,n} = \phi_\theta\left(\frac{t_n}{\varepsilon}, v_{\theta,n}, \varepsilon\right) + w_{\theta,n}, \tag{3.77}$$

Local truncation errors can be described in this way:

$$\begin{aligned} e_{\theta,n} &:= y^{\varepsilon}(t_n) - y^{\varepsilon}_{\theta,n} \\ &= \phi^{[p]}_{\frac{t_n}{\varepsilon}}(v(t_n)) + w(t_n) - \phi_\theta\left(\frac{t_n}{\varepsilon}, v_{\theta,n}, \varepsilon\right) - w_{\theta,n} \\ &= \phi^{[p]}_{\frac{t_n}{\varepsilon}}(v(t_n)) - \phi^{[p]}_{\frac{t_n}{\varepsilon}}(v_{\theta,n}) + \phi^{[p]}_{\frac{t_n}{\varepsilon}}(v_{\theta,n}) - \phi_\theta\left(\frac{t_n}{\varepsilon}, v_{\theta,n}, \varepsilon\right) + w(t_n) - w_{\theta,n}. \end{aligned} \tag{3.78}$$

The first difference term can be bounded from above using α_ϕ, the Lipschitz constant of $\phi^{[p]}$, and the second difference term can be estimated using the learning error for ϕ_θ. Thus we get

$$|e_{\theta,n}| \leqslant \alpha_\phi|e_{\theta,v,n}| + \delta_\phi\varepsilon + |e_{\theta,w,n}| \tag{3.79}$$

and the desired estimate.

3.C Appendix: Proof of Theorem 30

A standard proof used for autonomous ODE (consistency and local truncation error) gives, if we denote $t \longmapsto \psi^{\varepsilon}_t$, the solution associated with g^{ε} and $\psi^{\varepsilon}_{\theta,n} =$

$(\varphi_\theta(\cdot,h,\varepsilon))^n\,(y^\varepsilon(0))$:

$$\left|\psi^\varepsilon_{t_n}-\psi^\varepsilon_{\theta,n}\right|\leqslant\frac{e^{\lambda T}-1}{\lambda}\delta\varphi, \tag{3.80}$$

where λ is the Lipschitz constant of $R_{\theta,\varphi}$ with respect to y.

Then, using the following decomposition:

$$\begin{aligned} y^\varepsilon_{\theta,n}-y^\varepsilon(t_n) &= \phi_\theta\left(\frac{t_n}{\varepsilon},\psi^\varepsilon_{\theta,n},\varepsilon\right)-\phi^\varepsilon_{\frac{t_n}{\varepsilon}}\left(\psi^\varepsilon_{\theta,n}\right)\\ &+ \phi^\varepsilon_{\frac{t_n}{\varepsilon}}\left(\psi^\varepsilon_{\theta,n}\right)-\phi^\varepsilon_{\frac{t_n}{\varepsilon}}\left(\varphi^{g^\varepsilon}_{t_n}(y^\varepsilon(0))\right)\\ &+ \phi^\varepsilon_{\frac{t_n}{\varepsilon}}\left(\varphi^{g^\varepsilon}_{t_n}(y^\varepsilon(0))\right)-y^\varepsilon(t_n), \end{aligned} \tag{3.81}$$

Since the first term is bounded from above by δ_ϕ, the second term is bounded from above by $L\left|\psi^\varepsilon_{t_n}-\psi^\varepsilon_{\theta,n}\right|$, where L is the differential of $\phi^\varepsilon_{\frac{t_n}{\varepsilon}}$ w.r.t. space variable, and the third term is estimated by the exponential remainder, we get the desired estimate.

3.D Appendix: Implementation of implicit methods

In the formula (3.13), the numerical flow takes the input $\phi_{\theta,-}\left(\frac{t_0}{\varepsilon},y_0,\varepsilon\right)$. For implicit methods, however, the numerical flow is a function of both input and output. $\phi_{\theta,-}\left(\frac{t_0+h}{\varepsilon},y_1,\varepsilon\right)$ is considered as the output, and if we consider, for example, the midpoint rule, we have

$$\begin{aligned} \Phi_h^{F_\theta(\cdot,h,\varepsilon)}\left(\phi_{\theta,-}\left(\frac{t_0}{\varepsilon},y_0,\varepsilon\right)\right) &= \phi_{\theta,-}\left(\frac{t_0}{\varepsilon},y_0,\varepsilon\right)\\ &+ hF_\theta\left(\frac{\phi_{\theta,-}\left(\frac{t_0}{\varepsilon},y_0,\varepsilon\right)+\phi_{\theta,-}\left(\frac{t_0+h}{\varepsilon},y_1,\varepsilon\right)}{2},h,\varepsilon\right) \end{aligned} \tag{3.82}$$

3.E Appendix: Computation of learning errors

3.E.1 Space and time discretizations

To compute the learning error with respect to ε, we discretise the space and time domains. In the formulas (3.54) we make the following approximations:

$$\underset{y\in\Omega}{Max}\left|F_\theta\left(y,0,\varepsilon\right)-F^{[k]}(y)\right|\approx\underset{0\leqslant i\leqslant I}{Max}\left|F_\theta(y_i,\varepsilon)-F^{[k]}(y_i)\right| \tag{3.83}$$

and

$$\underset{(\tau,y)\in[0,2\pi]\times\Omega}{Max}\left|\phi_\theta\left(\tau,y,\varepsilon\right)-\phi_\tau^{[k]}(y)\right|\approx\underset{0\leqslant i\leqslant I,0\leqslant j\leqslant J}{Max}\left|\phi_\theta\left(\tau_j,y_i,\varepsilon\right)\right|. \tag{3.84}$$

where $\{y_i\}_{0\leqslant i\leqslant I}$ and $\{\tau_j\}_{0\leqslant j\leqslant J}$ are discretisations of Ω and $[0,2\pi]$ respectively. In our simulations we take $I=960=31^2-1$ ($\Omega=[-2,2]^2$ is represented by the points $\{(y_{1,l_1},y_{2,l_2})\}_{0\leqslant l_1,l_2\leqslant 30}$) and $J=30$.

3.E.2 Intergals and derivatives representation

We also take $\phi^{[0]}=Id$ and $F^{[0]}=\langle f\rangle$ for order 0. For order 1 we take, by the following formulae, (3.51) and (3.52):

$$\phi_\tau^{[1]}(y)=y+\varepsilon\int_0^\tau f(\sigma,y)\mathrm{d}\sigma \tag{3.85}$$

and

$$F^{[1]}(y)=\left(\frac{\partial\langle\phi_\cdot^{[1]}\rangle}{\partial y}(y)\right)^{-1}\left\langle f\left(\cdot,\phi_\cdot^{[1]}(y)\right)\right\rangle. \tag{3.86}$$

To compute an integral of the form $\int_a^b g(\sigma)\mathrm{d}\sigma$, we use a Gauss quadrature with 10 points.

Also, to approximate the space derivative of a function $g:\mathbb{R}^2\mapsto\mathbb{R}^2$ (Jacobian matrix), we use the following finite difference approximation:

$$\frac{\partial g}{\partial y}(y) = \frac{1}{2\eta}\left[g(y+\eta\cdot e_1) - g(y-\eta\cdot e_1) \quad g(y+\eta\cdot e_2) - g(y-\eta\cdot e_2)\right] + \mathcal{O}\left(\eta^2\right), \tag{3.87}$$

where e_1, e_2 are the two vectors of the canonical base of $\mathbb{R}^2$.

Furthermore, if $g, F = \mathbb{R}^d \longrightarrow \mathbb{R}^d$, we can approximate the directional derivative $\frac{\partial g}{\partial y}(y)F(y)$ using this finite difference approximation:

$$\frac{\partial g}{\partial y}(y)F(y) = \frac{1}{2\eta}\Big[g(y+\eta F(y) - g(y-\eta F(y)\Big] + \mathcal{O}(\eta^2). \tag{3.88}$$

We take $\eta = 10^{-5}$ in our case.

3.F Appendix: Influence of learning error

We assess the impact of learning error on integration error, as discussed in Theorems 28, 29, and 30. To illustrate this property, we compare UA tests after two trainings using different numbers of hidden layers and neurons with Micro-macro correction. Figure 3.33 demonstrates that efficient learning leads to a reduction in integration error.

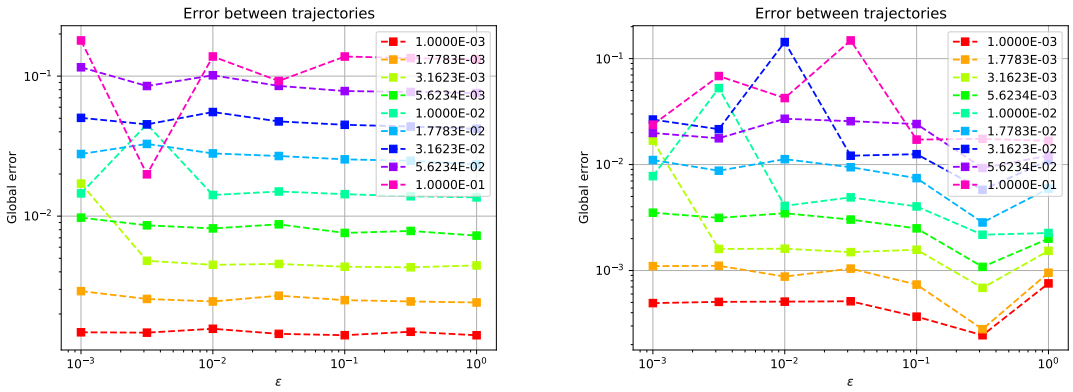

Figure 3.33 – Influence of learning error over integration error (UA test) for Van der Pol oscillator. Left: $K = 800$ data, 25 neurons and 1 hidden layer per neural network. Right: $K = 500\,000$ data, 150 neurons and 2 hidden layers per neural network.

3.G Appendix: Choice of the parameters

3.G.1 Inverted Pendulum - Forward Euler method

Parameters	
# Math Parameters:	
Dynamical system:	Inverted Pendulum
Numerical method:	Forward Euler
Interval where time steps are selected:	$[h_-, h_+] = [10^{-3}, 10^{-1}]$
Interval where small parameters are selected:	$[\varepsilon_-, \varepsilon_+] = [10^{-3}, 1]$
Time for ODE simulation:	$T = 1$
Time step for ODE simulation:	$h = 0.01$
High oscillation parameter for ODE simulation:	$\varepsilon \in \left\{5 \cdot 10^{-2}, 10^{-3}\right\}$
Initial datum:	$y^\varepsilon(0) = (0.5, -0.5)$
# Machine Learning Parameters:	
Domain where initial data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 1\,000\,000$
Proportion of data for training:	80% - $K_0 = 800\,000$
Batch size:	$B = 100$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Epochs:	200

Computational time for data creation: 1 h 44 min 19 s
Computational time for training: 9 h 33 min 24s

3.G.2 Inverted Pendululm - midpoint method

Parameters	
# Math Parameters:	
Dynamical system:	Inverted Pendulum
Numerical method:	midpoint
Interval where time steps are selected:	$[h_-, h_+] = [10^{-3}, 10^{-1}]$
Interval where small parameters are selected:	$[\varepsilon_-, \varepsilon_+] = [10^{-3}, 1]$
Time for ODE simulation:	$T = 1$
Time step for ODE simulation:	$h = 0.01$
High oscillation parameter for ODE simulation:	$\varepsilon \in \left\{5 \cdot 10^{-2}, 10^{-3}\right\}$
Initial datum:	$y^\varepsilon(0) = (0.5, -0.5)$
# Machine Learning Parameters:	
Domain where initial data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 1\,000\,000$
Proportion of data for training:	80% - $K_0 = 800\,000$
Batch size:	$B = 100$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Epochs:	200

Computational time for data creation: 1 h 44 min 19 s [4]
Computational time for training: 9 h 11 min 8 s

4. data set used for Inverted Pendulum is the same for both Forward Euler and midpoint.

3.G.3 Van der Pol oscillator - Forward Euler method

Parameters	
# Math Parameters:	
Dynamical system:	VDP
Numerical method:	Forward Euler
Interval where time steps are selected:	$[h_-, h_+] = [10^{-3}, 10^{-1}]$
Interval where small parameters are selected:	$[\varepsilon_-, \varepsilon_+] = [10^{-3}, 1]$
Time for ODE simulation:	$T = 1$
Time step for ODE simulation:	$h = 0.01$ (for $\varepsilon = 0.1$) and $h = 0.001$ (for $\varepsilon = 0.01$)
High oscillation parameter for ODE simulation:	$\varepsilon \in \left\{\cdot 10^{-1}, 10^{-2}\right\}$
Initial datum:	$y^\varepsilon(0) = (0.5, 0.5)$
# Machine Learning Parameters:	
Domain where initial data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 20\,000\,000$
Proportion of data for training:	80% - $K_0 = 16\,000\,000$
Batch size:	$B = 200$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Epochs:	200

Computational time for data creation: 2 Days 8 h 2 min 30 s
Computational time for training: 3 Days 13 h 50 min 30 s

3.G.4 Van der Pol oscillator: Comparison between classical and alternative method

Parameters	
# Math Parameters:	
Dynamical system:	VDP
Interval where time steps are selected:	$[h_-, h_+] = [10^{-3}, 10^{-1}]$
Interval where small parameters are selected:	$[\varepsilon_-, \varepsilon_+] = [10^{-3}, 0.2]$
Time for ODE simulation:	$T = 1$
Time step for ODE simulation:	$h = 0.001$
Small parameter for ODE simulation:	$\varepsilon \in \left\{10^{-3}, 10^{-2}, 10^{-1}\right\}$
Initial datum:	$y^\varepsilon(0) = (0.5, 0.5)$
# Machine Learning Parameters:	
Domain where initial data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 100\,000$
Proportion of data for training:	80% - $K_0 = 80\,000$
Batch size:	$B = 100$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Epochs:	200

- **Classical method (with auto-encoder):**
  Computational time for data creation: 10 min 48 s
  Computational time for training: 57 min 03 s

- **Alternative method adapted for autonomous case:**
  Computational time for data creation: 10 min 52 s
  Computational time for training: 48 min 47 s

Chapter 4

PINN's for highly oscillatory equations

In this chapter, we will give a method based on Physics Informed Neural Networks (PINN's) in order to approximate solutions of highly oscillatory differential equations of the form (20).

4.1 Introduction

PINN's are widely used in order to solve ordinary or partial differential equations [64, 74, 103, 115], and especially multiscale problems [60, 109] by inserting structure of the equation and additionnal properties in the *Loss* function, such as multiscale, boundary and initial conditions.

The main idea of this chapter is to use the multi-scale structure of the solution of an highly oscillatory ODE [24, 87] in order to model it with neural networks as in the previous chapter and insert it in the PINN. Thus, the solution is directly learned from then equation.

4.2 Solving highly oscillatory ODE's with PINN's

In order to solve an highly oscillatory ODE of the form (20), we mimic the equation by considering the PINN as an approximation of the exact solution, and we take into consideration initial or (for PDE's) boundary conditions.

4.2.1 General strategy

In order to solve an equation of the form

$$\left\{ \begin{array}{rcl} \dot{y^\varepsilon}(t) & = & f\left(\frac{t}{\varepsilon}, y^\varepsilon(t)\right) \\ y^\varepsilon(0) & \in & \mathbb{R}^d \end{array} \right. \tag{4.1}$$

by using a PINN's, we consider an approximation of the exact flow $(t,x) \longmapsto \varphi^f_{0,t}(x)$ with a neural network $(t,x) \longmapsto \varphi^f_\theta(t,x)$. Thus, if we take into account equation (20), we have the following approximations:

$$\left\{ \begin{array}{rcl} \frac{\partial \varphi^f_\theta}{\partial t}(t,x) & \approx & f\left(\frac{t}{\varepsilon}, \varphi^f_\theta(t,x)\right) \\ \varphi^f_\theta(0,x) & \approx & x \in \mathbb{R}^d \end{array} \right. \tag{4.2}$$

Therefore, we have to train the neural network φ^f_θ in order to educe the error generated by the approximations.

4.2.2 Machine Learning method

The main idea of the strategy is to reduce the error generated by approximations in equation (4.2). We want to approximate exact flow $(t,x) \longmapsto \varphi^f_{0,t}(x)$ for all $(t,x) \in [0,t] \times \Omega$, where $\Omega \subset \mathbb{R}^d$ is a compact set. We chose the L^2-error in order to represent the approximations errors, which is given by

$$E_{L^2} := \int_{[0,T]\times\Omega} \left| \frac{\partial \varphi^f_\theta}{\partial t}(t,x) - f\left(\frac{t}{\varepsilon}, \varphi^f_\theta(t,x)\right) \right|^2 \mathrm{d}t\mathrm{d}x + \int_\Omega \left|\varphi^f_\theta(0,x) - x\right|^2 \mathrm{d}x. \tag{4.3}$$

First term of E_{L^2} corresponds to the learning of the dynamics of the ODE whereas second term corresponds to the learning of initial conditions.

1. **Discretization of the domain:** This step replaces the data set construction. We randomly select K points $(t_k, x_k) \in [0,T] \times \Omega$ for all $k \in [\![0, K-1]\!]$.

2. **Training of the neural network:** We minimize the MSE $Loss_{Train}$ function, which is a space-time discretization of the L^2-error E_{L^2} over the K_0 first points. We compute optimal neural network by using a gradient method:

$$\begin{aligned} Loss_{Train} &= \frac{\xi}{K_0}\sum_{k=0}^{K_0-1}\left|\frac{\partial\varphi_\theta^f}{\partial t}(t_k,x_k) - f\left(\frac{t_k}{\varepsilon},\varphi_\theta^f(t_k,x_k)\right)\right|^2 \\ &+ \frac{1}{K_0}\sum_{k=0}^{K_0-1}\left|\varphi_\theta^f(0,x_k)-x_k\right|^2 \end{aligned} \tag{4.4}$$

where coefficient $\xi \geqslant 0$ is a weight, in order to give more or fewer importance to initial conditions learning or to give the same rough size to both terms in order to get a similar learning error for both equation and initial condition. K_0 corresponds to the number of data selected for training.

At the same time, we compute the value of $Loss_{Test}$, which evaluated the learning error over the remaining data which are not used for training:

$$\begin{aligned} Loss_{Test} &= \frac{\xi}{K-K_0}\sum_{k=K_0}^{K-1}\left|\frac{\partial\varphi_\theta^f}{\partial t}(t_k,x_k) - f\left(\frac{t_k}{\varepsilon},\varphi_\theta^f(t_k,x_k)\right)\right|^2 \\ &+ \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\left|\varphi_\theta^f(0,x_k)-x_k\right|^2 \end{aligned} \tag{4.5}$$

If $Loss_{Train}$ and $Loss_{Test}$ have the same decay order, we get an accurate learning of both dynamics of the equation and initial conditions.

3. **Numerical approximation:** After training, we select an initial condition $y_0 \in \Omega$ and we plot the approximated solution $t \longmapsto \varphi_\theta^f(t,y_0)$ for all $t \in [0,T]$.

4.2.3 Neural Network structure

In order to approximate the solution $t \longmapsto \varphi_{0,t}^f(y_0)$ of the equation, we mimic the *slow-fast* decomposition formula (3.2) with Neural networks

$$\varphi_\theta^f(t,y) = \phi_\theta\left(\frac{t}{\varepsilon},\psi_\theta(t,y)\right), \tag{4.6}$$

where ψ_θ is a Multi-Layer Perceptron and ϕ_θ is given by

$$\phi_\theta(\tau,y) = y + \varepsilon\left[R_\theta^\phi(\cos(\tau),\sin(\tau),y) - R_\theta^\phi(1,0,y)\right] \tag{4.7}$$

where R_θ^ϕ is a MLP too. With this structure, *slow-fast* decomposition and its properties is preserved.

4.2.4 Error analysis

Now, using learning errors, we can give an error estimate over the approximated solution.

Theorem 31. *Let consider $\varepsilon \in]0,1]$ the high oscillation parameter. For all $(t,x) \in [0,T] \times \Omega$, let consider the approximation error at time t for initial condition x*

$$e(t,x) := \varphi_\theta^f(t,x) - \varphi_{0,t}^f(x). \tag{4.8}$$

Let denote the following learning errors:

(i) Dynamics of the equation:

$$\delta_{ODE} := \underset{(t,x)\in[0,T]\times\Omega}{Max} \left| \frac{\partial \varphi_\theta^f}{\partial t}(t,x) - f\left(\frac{t}{\varepsilon}, \varphi_\theta^f(t,x)\right) \right| \tag{4.9}$$

(ii) Initial condition:

$$\delta_{IC} := \underset{x\in\Omega}{Max} \left| \varphi_\theta^f(0,x) - x \right| \tag{4.10}$$

Then there exists a constant $\tilde{L}_f > 0$ (independant from T) such that, for all $t \in [0,T]$:

$$|e(t,x)| \leqslant (\delta_{IC} + T\delta_{ODE})\, e^{\tilde{L}_f T} \tag{4.11}$$

4.3 Numerical experiments

In order to illustrate using of PINN's, method described in section 4.2 is tested over a dynamical system.

Hénon-Heiles system: This is an hamiltonian system which describes the motion of a star around a galactic center. This system is governed by the following system

$$\left\{\begin{array}{lcl} \dot{q_1} &=& \frac{1}{\varepsilon}p_1 \\ \dot{q_2} &=& \frac{1}{\varepsilon}p_2 \\ \dot{p_1} &=& -q_1 - 2q_1q_2 \\ \dot{p_2} &=& -q_2 - q_1^2 + \frac{3}{2}q_2^2. \end{array}\right. \tag{4.12}$$

Notice that this system is an Hamiltonian system, with associated hamiltonian given by $H : (q_1, q_2, p_1, p_2) \longmapsto \frac{1}{2\varepsilon}(p_1^2 + p_2^2) + \frac{1}{2}(q_1^2 + q_2^2) + q_1^2q_2 - \frac{1}{2}q_2^3$.

If we make the following variable change

$$\left\{\begin{array}{lcl} \dot{y_1^\varepsilon}(t) &=& \cos\left(\frac{t}{\varepsilon}\right) q_1(t) - \sin\left(\frac{t}{\varepsilon}\right) p_1(t) \\ \dot{y_2^\varepsilon}(t) &=& q_2(t) \\ \dot{y_3^\varepsilon}(t) &=& \sin\left(\frac{t}{\varepsilon}\right) q_1(t) + \cos\left(\frac{t}{\varepsilon}\right) p_1(t) \\ \dot{y_4^\varepsilon}(t) &=& p_2(t), \end{array}\right. \tag{4.13}$$

then we obtain the next system, under its following form:

$$\left\{\begin{array}{lcl} \dot{y_1^\varepsilon}(t) &=& 2\sin\left(\frac{t}{\varepsilon}\right)\left[\cos\left(\frac{t}{\varepsilon}\right) y_1^\varepsilon(t) + \sin\left(\frac{t}{\varepsilon}\right) y_3^\varepsilon(t) + \sin\left(\frac{t}{\varepsilon}\right)\right] y_2^\varepsilon(t) \\ \dot{y_2^\varepsilon}(t) &=& y_4^\varepsilon(t) \\ \dot{y_3^\varepsilon}(t) &=& -2\cos\left(\frac{t}{\varepsilon}\right)\left[\cos\left(\frac{t}{\varepsilon}\right) y_1^\varepsilon(t) + \cos\left(\frac{t}{\varepsilon}\right) y_3^\varepsilon(t) + \sin\left(\frac{t}{\varepsilon}\right)\right] y_2^\varepsilon(t) \\ \dot{y_4^\varepsilon}(t) &=& -\left[\cos\left(\frac{t}{\varepsilon}\right) y_1^\varepsilon(t) + \sin\left(\frac{t}{\varepsilon}\right) y_3^\varepsilon(t)\right]^2 + \frac{3}{2}y_2^\varepsilon(t)^2 - y_2^\varepsilon(t), \end{array}\right. \tag{4.14}$$

which corresponds to the canonical form for highly oscillatory equations.

Van der Pol oscillator: This is a system which describes an electric system with nonlinear damping. It is governed by the following two-dimensional system

$$\left\{\begin{array}{lcl} \dot{q} &=& \frac{1}{\varepsilon}p \\ \dot{p} &=& -\frac{1}{\varepsilon}q\left(\frac{1}{4} - q^2\right)p \end{array}\right. \tag{4.15}$$

by making the variable change $(y_1, y_2) \longmapsto S\left(\frac{t}{\varepsilon}\right)(q, p)$, where $\tau \longmapsto S(\tau)$ is given by

$$\tau \longmapsto S(\tau) = \begin{bmatrix} \cos(\tau) & -\sin(\tau) \\ \sin(\tau) & \cos(\tau) \end{bmatrix} \tag{4.16}$$

we get the system:

$$\begin{cases} \dot{y_1}(t) &= -\sin\left(\frac{t}{\varepsilon}\right)\left[\frac{1}{4} - \left(y_1(t)\cos\left(\frac{t}{\varepsilon}\right) + y_2(t)\sin\left(\frac{t}{\varepsilon}\right)\right)^2\right]\left[-y_1(t)\sin\left(\frac{t}{\varepsilon}\right) + y_2(t)\cos\left(\frac{t}{\varepsilon}\right)\right] \\ \dot{y_2}(t) &= \cos\left(\frac{t}{\varepsilon}\right)\left[\frac{1}{4} - \left(y_1(t)\cos\left(\frac{t}{\varepsilon}\right) + y_2(t)\sin\left(\frac{t}{\varepsilon}\right)\right)^2\right]\left[-y_1(t)\sin\left(\frac{t}{\varepsilon}\right) + y_2(t)\cos\left(\frac{t}{\varepsilon}\right)\right] \end{cases} \tag{4.17}$$

which corresponds to the canonical form for highly oscillatory systems.

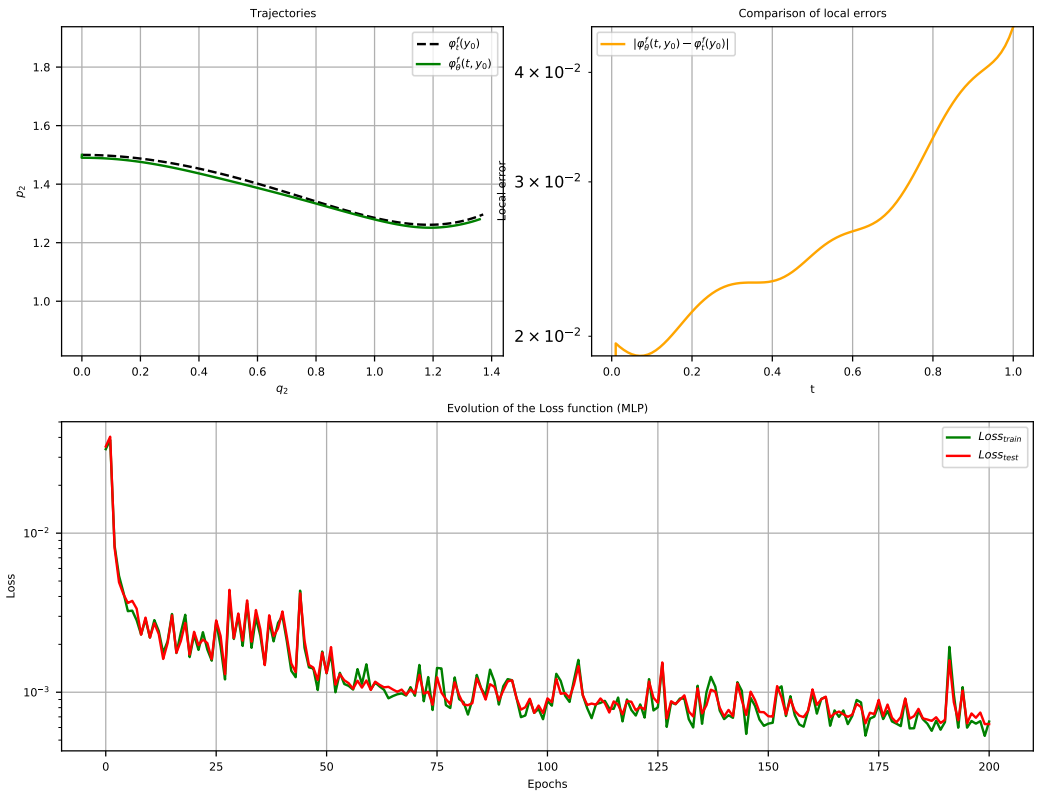

Figure 4.1 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow $t \longmapsto \varphi^f_{0,t}(y_0)$, green: learned flow $t \longmapsto \varphi^f_\theta(t, y_0)$ and local error (yellow) for the Hénon-Heiles system in the case $\varepsilon = 1$ and $T = 1$.

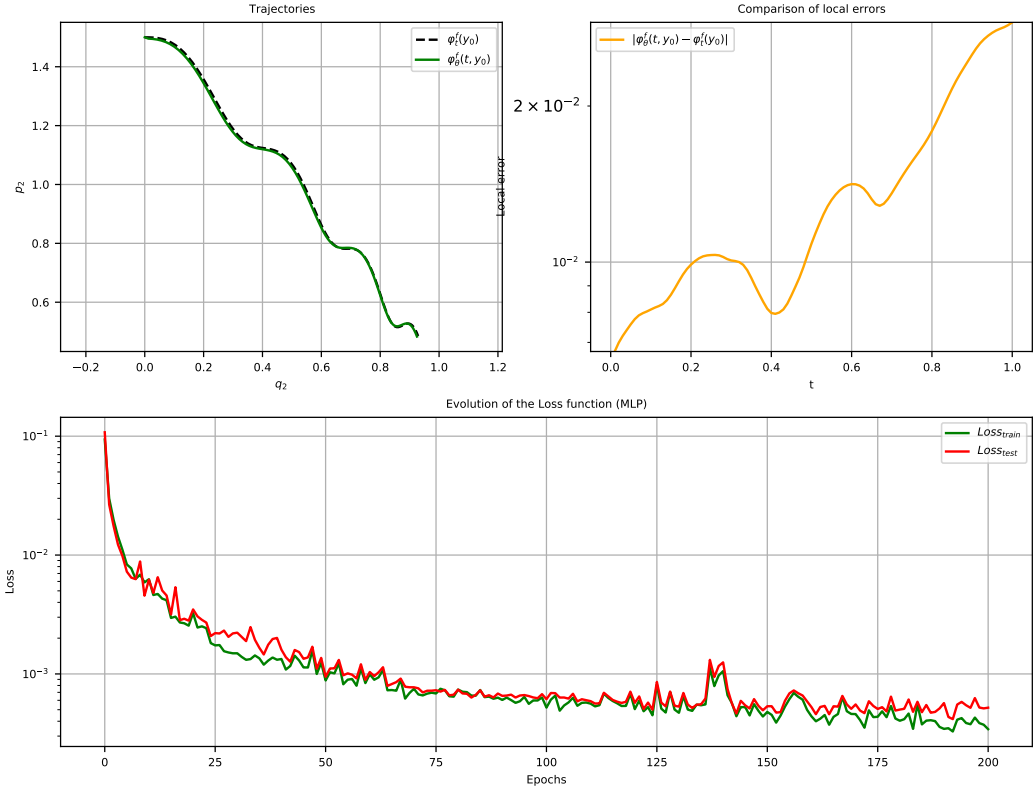

Figure 4.2 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow $t \longmapsto \varphi^f_{0,t}(y_0)$, green: learned flow $t \longmapsto \varphi^f_\theta(t, y_0)$ and local error (yellow) for the Hénon-Heiles system in the case $\varepsilon = 0.1$ and $T = 1$.

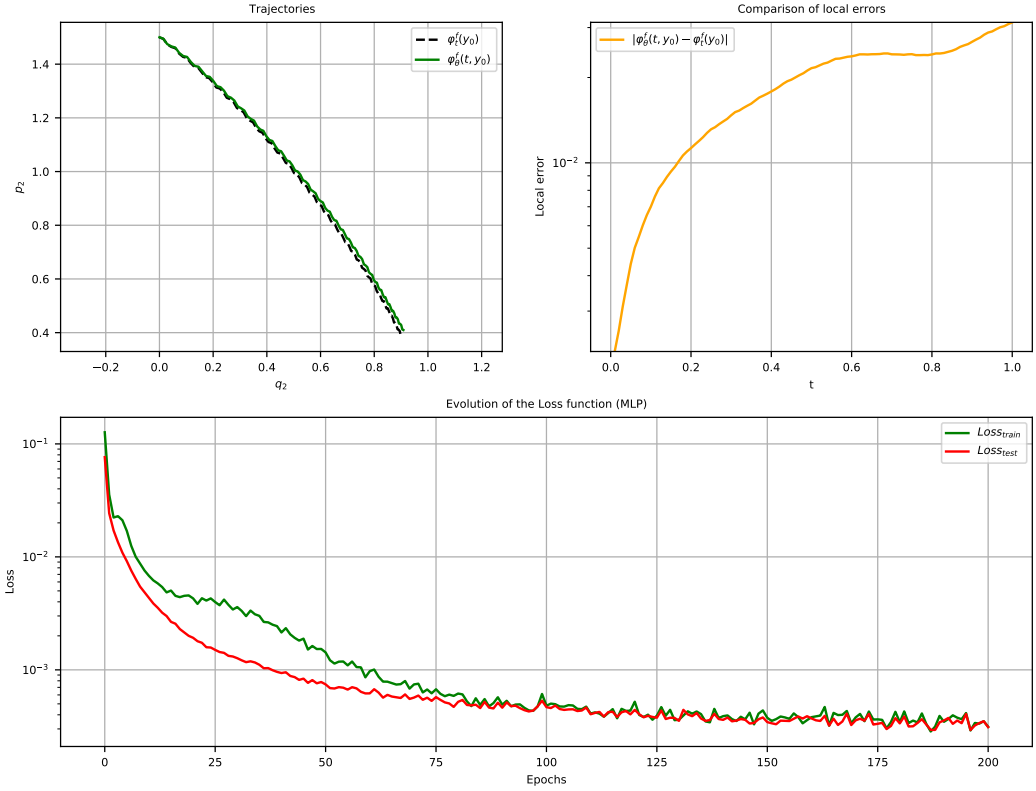

Figure 4.3 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow $t \longmapsto \varphi^f_{0,t}(y_0)$, green: learned flow $t \longmapsto \varphi^f_\theta(t, y_0)$ and local error (yellow) for the Hénon-Heiles system in the case $\varepsilon = 0.01$ and $T = 1$.

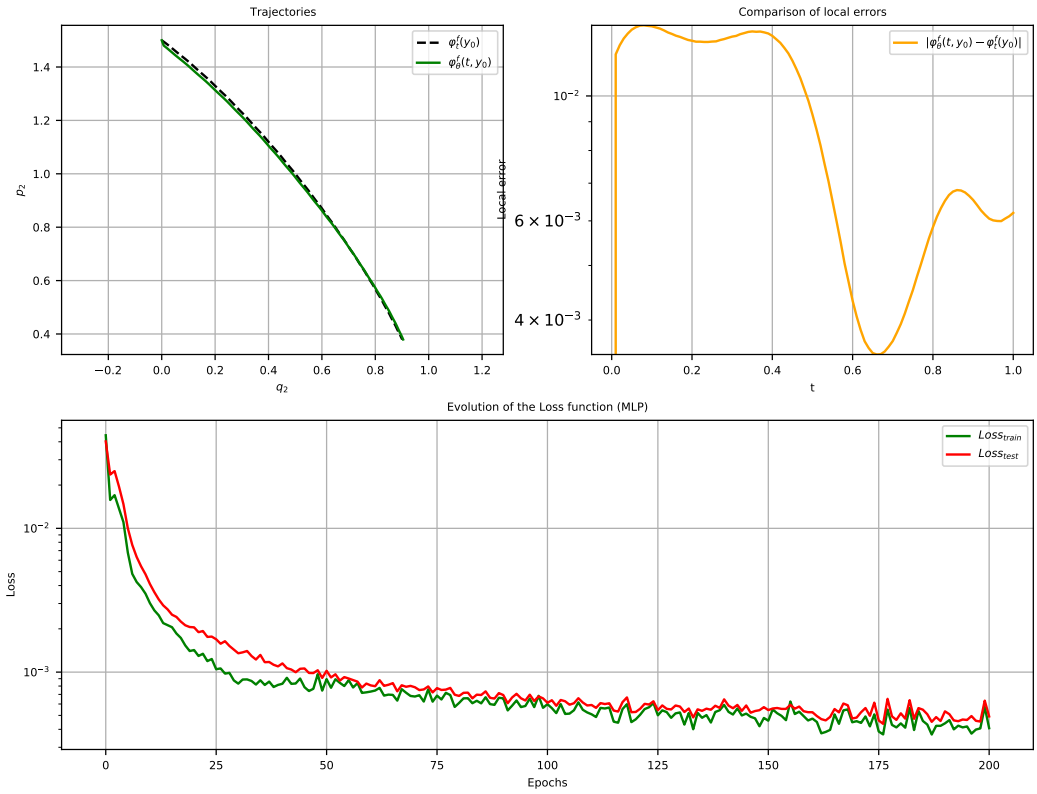

Figure 4.4 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow $t \longmapsto \varphi^f_{0,t}(y_0)$, green: learned flow $t \longmapsto \varphi^f_\theta(t, y_0)$ and local error (yellow) for the Hénon-Heiles system in the case $\varepsilon = 0.001$ and $T = 1$.

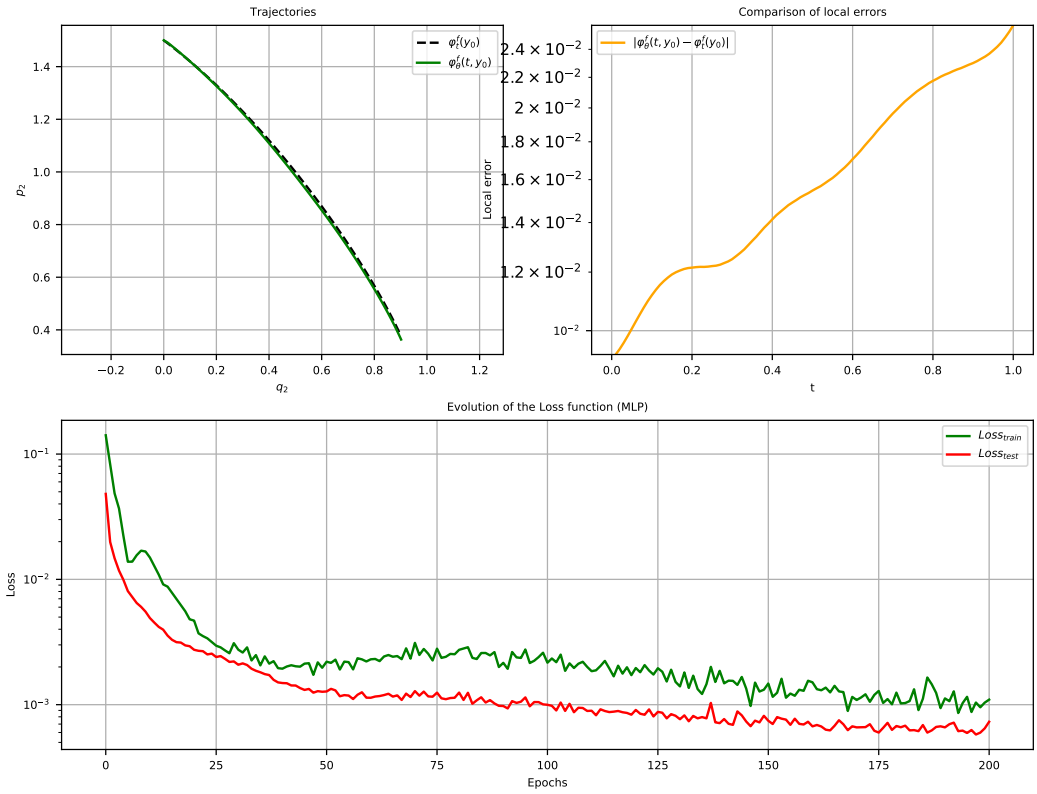

Figure 4.5 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow $t \longmapsto \varphi^f_{0,t}(y_0)$, green: learned flow $t \longmapsto \varphi^f_\theta(t, y_0)$ and local error (yellow) for the Hénon-Heiles system in the case $\varepsilon = 10^{-5}$ and $T = 1$.

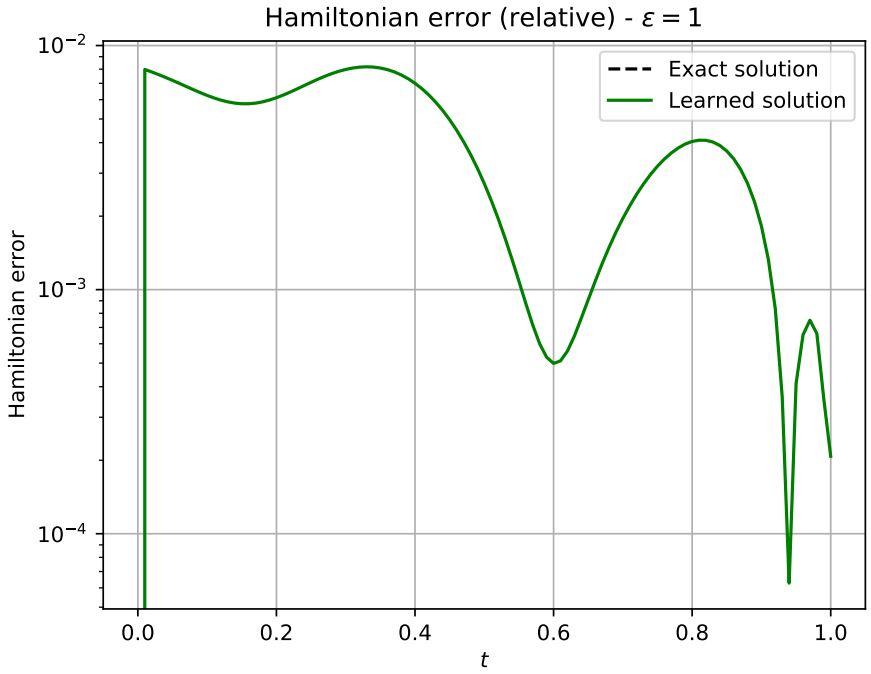

Figure 4.6 – Hamiltonian error over the function $(t \longmapsto (q_\theta(t), p_\theta(t))$, obtained by applying the inverse variable change (4.13) to the function $t \longmapsto \varphi_\theta^f(t, y_0)$ in the case $\varepsilon = 1$.

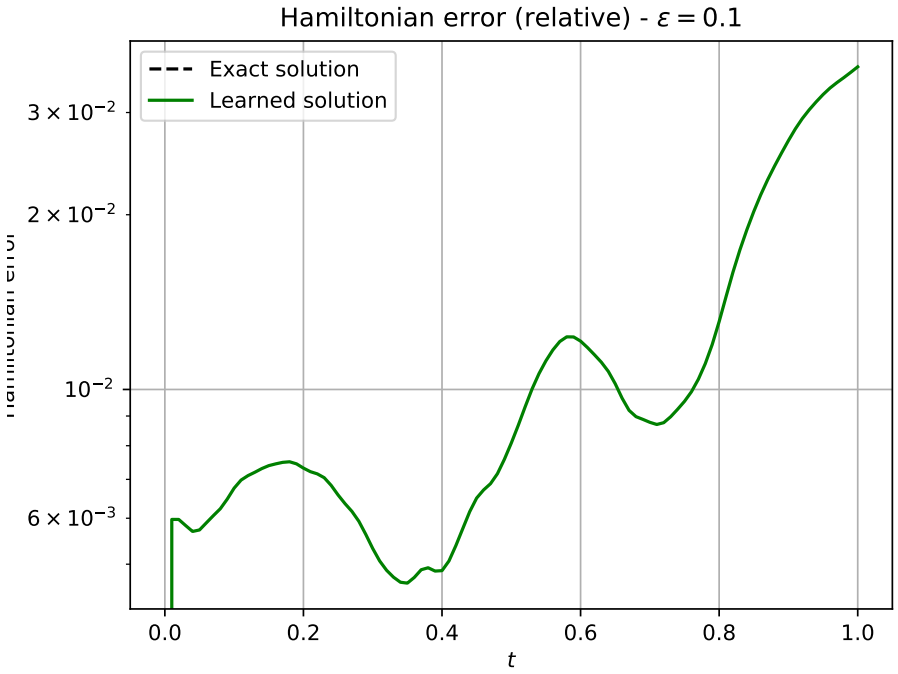

Figure 4.7 – Hamiltonian error over the function $(t \longmapsto (q_\theta(t), p_\theta(t))$, obtained by applying the inverse variable change (4.13) to the function $t \longmapsto \varphi_\theta^f(t, y_0)$ in the case $\varepsilon = 0.1$.

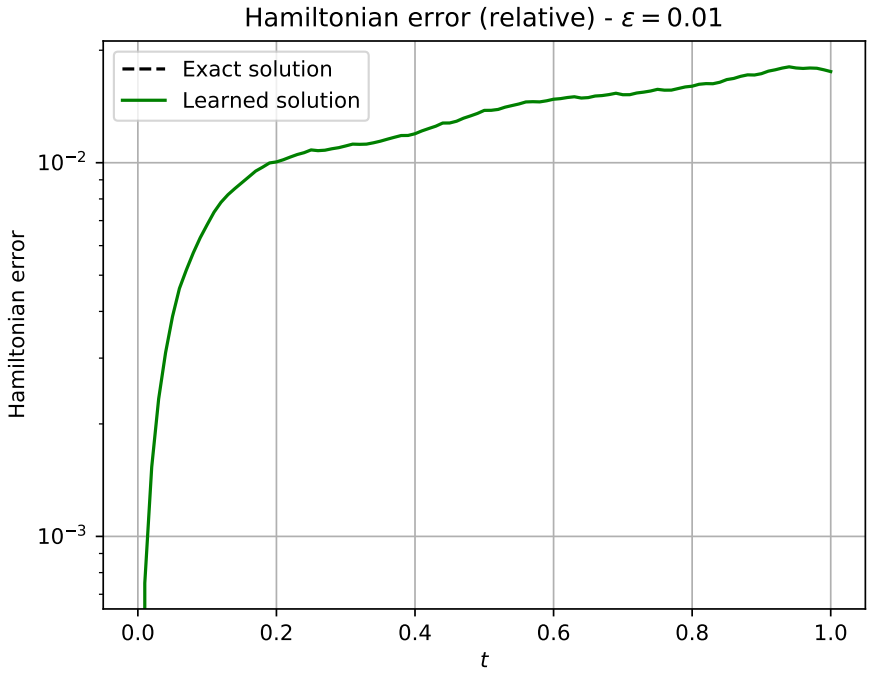

Figure 4.8 – Hamiltonian error over the function $(t \longmapsto (q_\theta(t), p_\theta(t))$, obtained by applying the inverse variable change (4.13) to the function $t \longmapsto \varphi_\theta^f(t, y_0)$ in the case $\varepsilon = 0.01$.

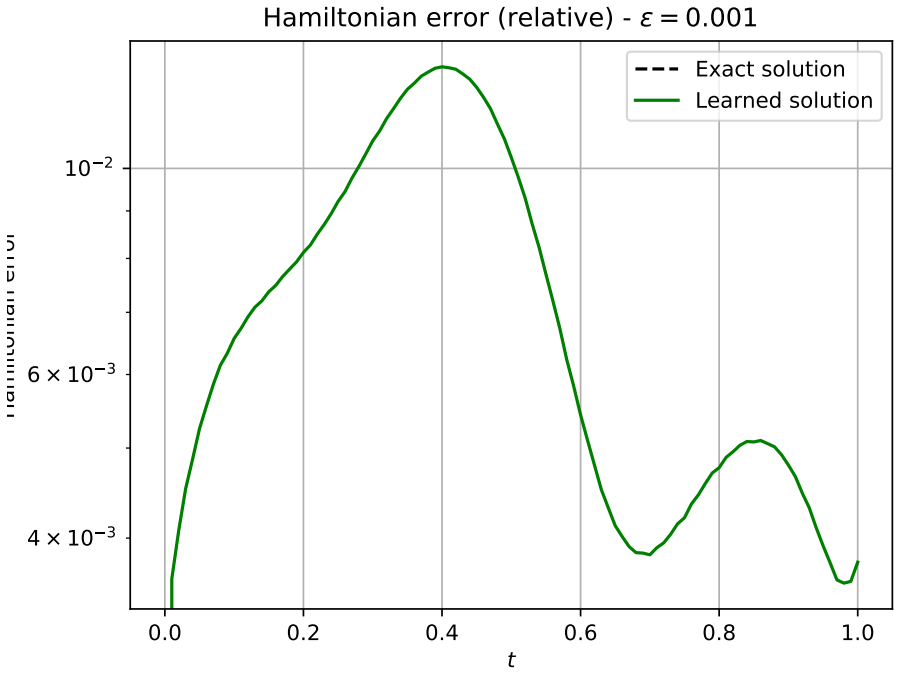

Figure 4.9 – Hamiltonian error over the function $(t \longmapsto (q_\theta(t), p_\theta(t))$, obtained by applying the inverse variable change (4.13) to the function $t \longmapsto \varphi_\theta^f(t, y_0)$ in the case $\varepsilon = 0.001$.

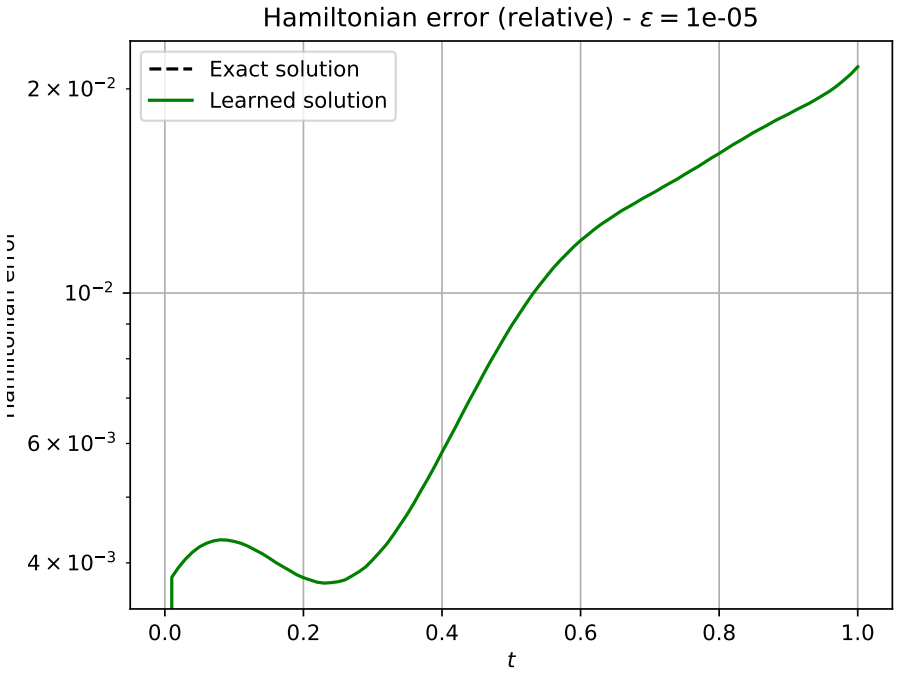

Figure 4.10 – Hamiltonian error over the function $(t \longmapsto (q_\theta(t), p_\theta(t))$, obtained by applying the inverse variable change (4.13) to the function $t \longmapsto \varphi_\theta^f(t, y_0)$ in the case $\varepsilon = 10^{-5}$.

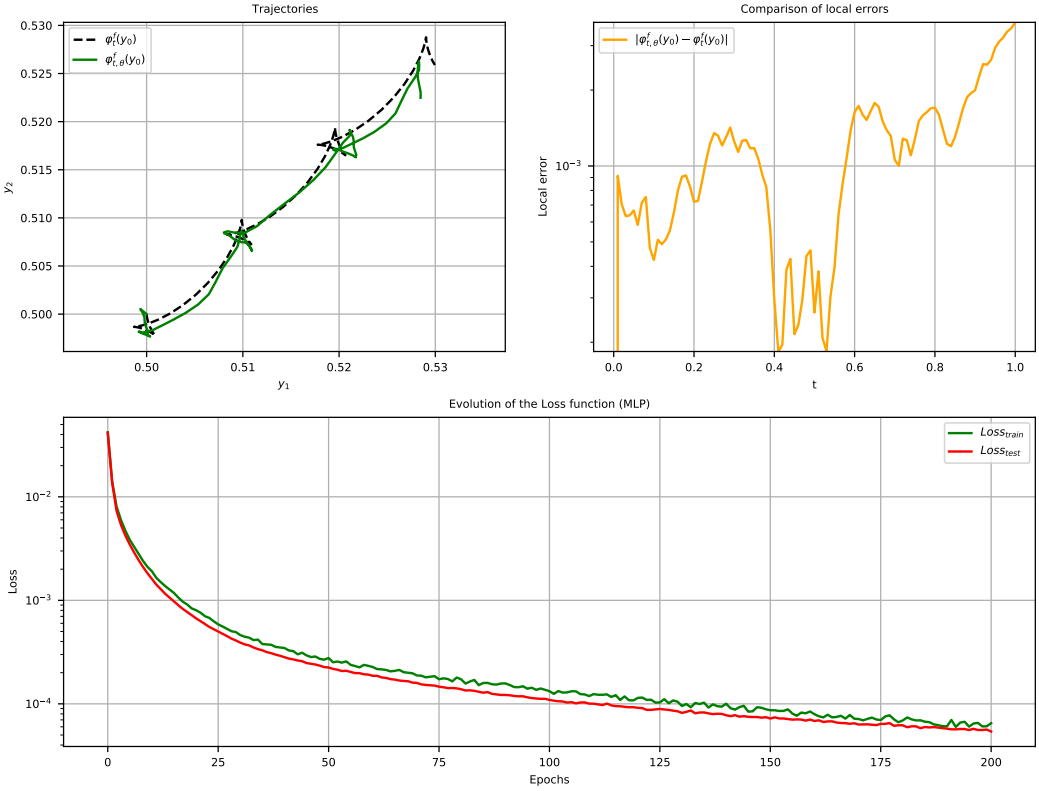

Figure 4.11 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow $t \longmapsto \varphi^f_{0,t}(y_0)$, green: learned flow $t \longmapsto \varphi^f_\theta(t, y_0)$ and local error (yellow) for the Van der Pol oscillator in the case $\varepsilon = 0.1$ and $T = 1$.

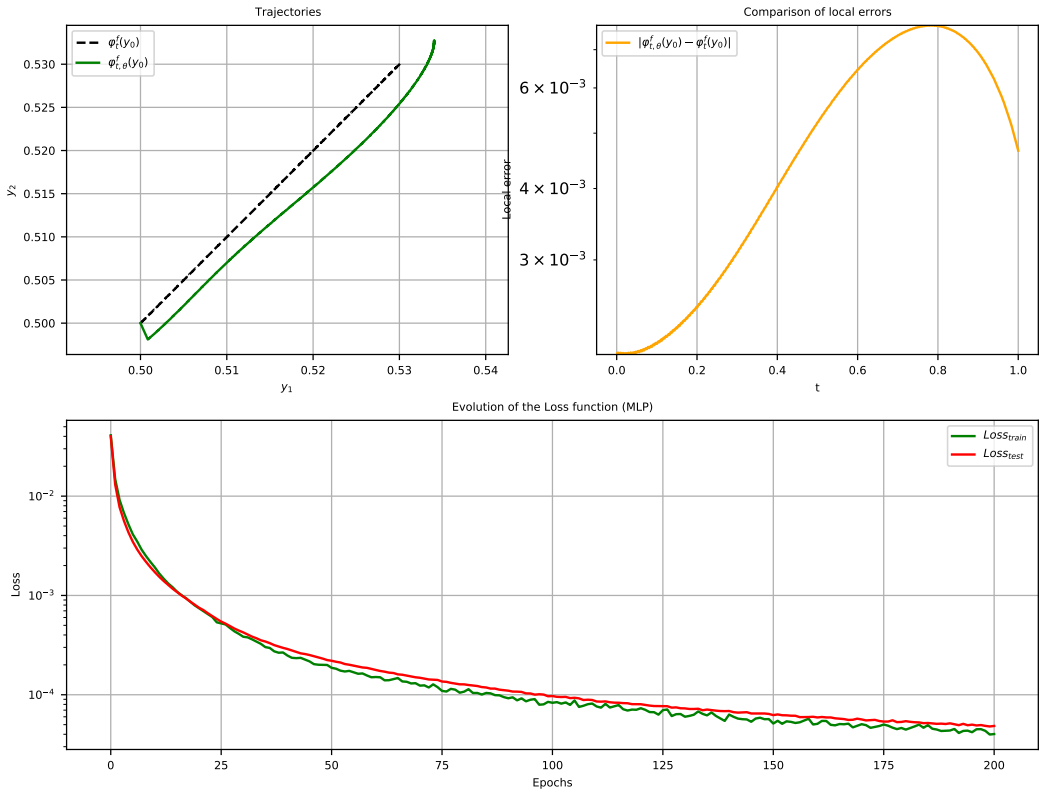

Figure 4.12 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow $t \longmapsto \varphi^f_{0,t}(y_0)$, green: learned flow $t \longmapsto \varphi^f_\theta(t, y_0)$ and local error (yellow) for the Van der Pol oscillator in the case $\varepsilon = 0.001$ and $T = 1$.

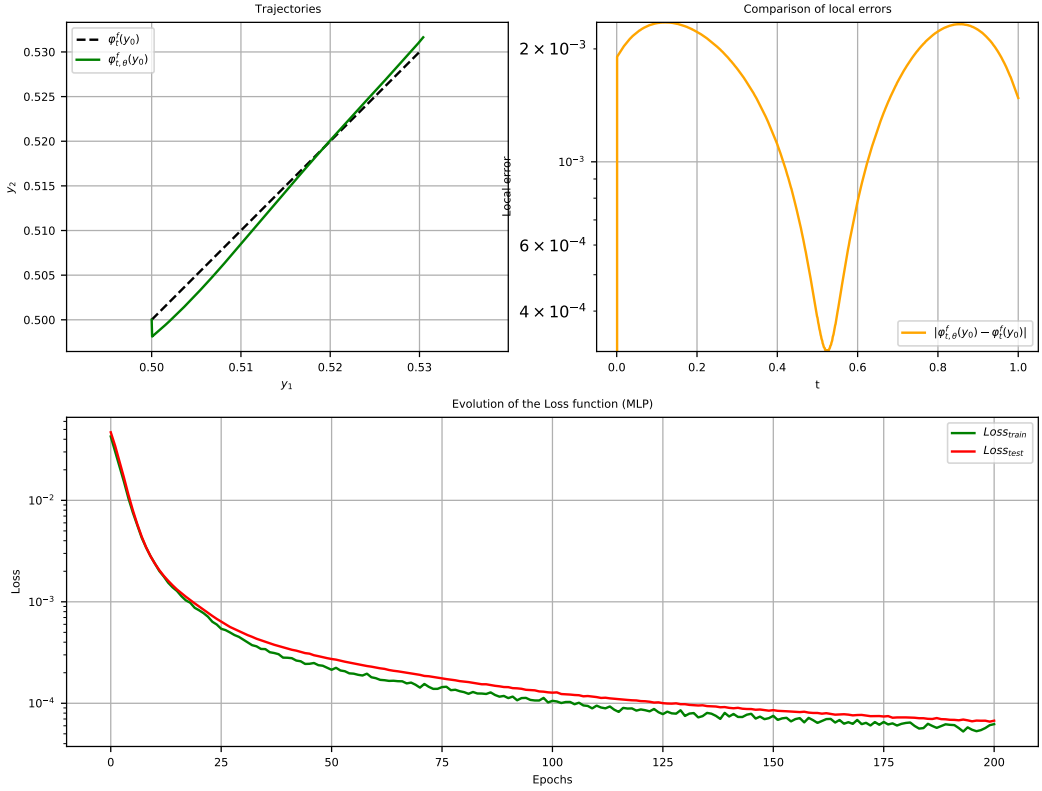

Figure 4.13 – Comparison between *Loss* decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow $t \longmapsto \varphi^f_{0,t}(y_0)$, green: learned flow $t \longmapsto \varphi^f_\theta(t, y_0)$ and local error (yellow) for the Van der Pol oscillator in the case $\varepsilon = 10^{-5}$ and $T = 1$.

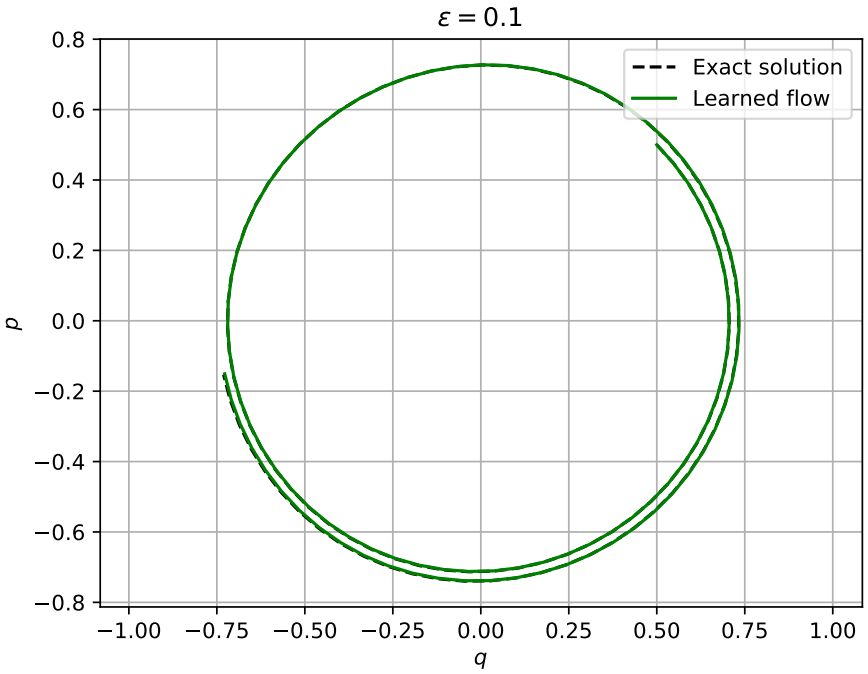

Figure 4.14 – Comparison between trajectories (dashed dark: exact flow, green: numerical flow with learned vector field) after the inverse variable change in the case $\varepsilon = 0.1$

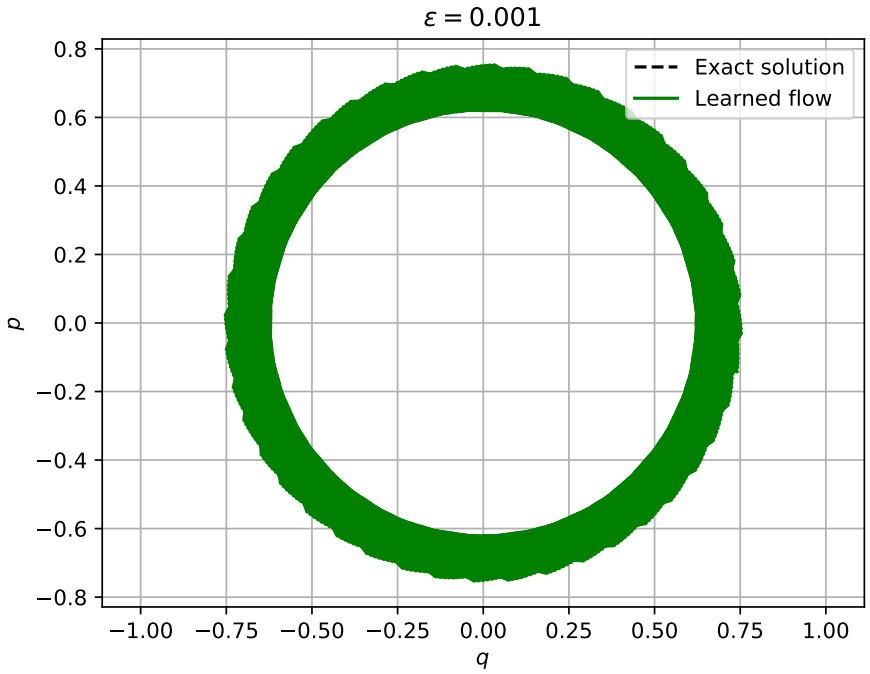

Figure 4.15 – Comparison between trajectories (dashed dark: exact flow, green: numerical flow with learned vector field) after the inverse variable change in the case $\varepsilon = 0.001$.

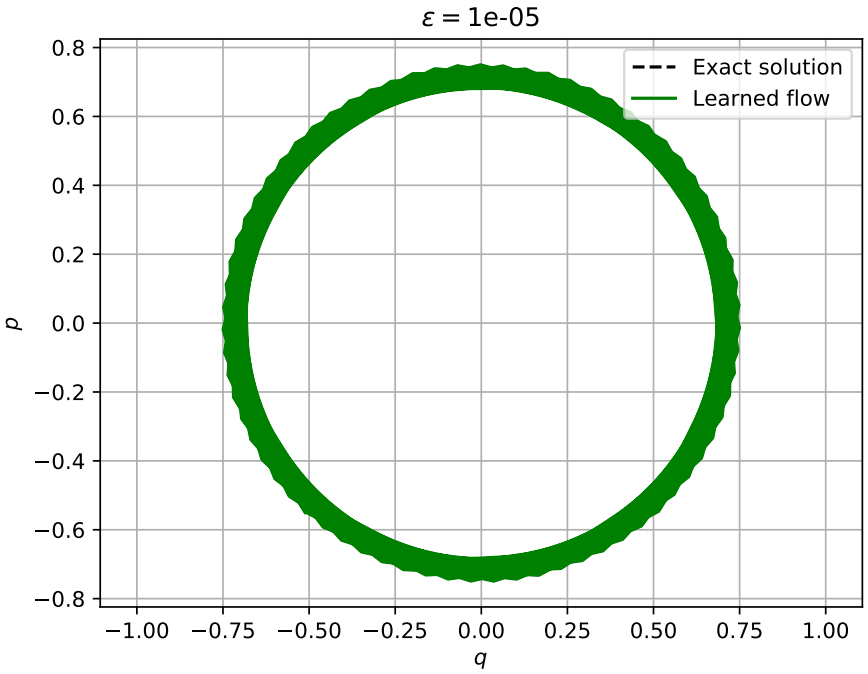

Figure 4.16 – Comparison between trajectories (dashed dark: exact flow, green: numerical flow with learned vector field) after the inverse variable change in the case $\varepsilon = 10^{-5}$.

4.4 Conclusions

In this chapter, we got an additional method to approximate solution of an highly oscillatory equation. Moreover, this method is not based over an existing numerical scheme as in previous chapters, and allows us to directly learn the solution from the equation, by only following the structure of the solution. However, this method is not consistent (we can't reduce the numerical error by decreasing the step size, which is only used to plot the numerical solution), and a new training has to be done for each new value of ε. Eventually, it can be complicated to train the network for values of ε close to 1 due to the structure of the solution and exponential remainder.

4.A Appendix: Proof of Theorem 31

Let $x \in \Omega$ and $t \in [0, T]$. We have

$$e(t,x) = e(0,x) + \int_0^t \frac{\partial e}{\partial t}(s,x)\mathrm{d}s. \tag{4.18}$$

By using definition of approximation error, we get

$$\begin{aligned} e(t,x) &= e(0,x) + \int_0^t \frac{\partial \varphi_\theta^f}{\partial t}(s,x) - \frac{\partial \varphi_{0,s}}{\partial t}(x)\mathrm{d}s. \\ &= e(0,x) + \int_0^t \frac{\partial \varphi_\theta^f}{\partial t}(s,x) - f\left(\frac{s}{\varepsilon}, \varphi_\theta^f(s,x)\right)\mathrm{d}s \\ &+ \int_0^t f\left(\frac{s}{\varepsilon}, \varphi_\theta^f(s,x)\right) - f\left(\frac{s}{\varepsilon}, \varphi_{0,s}^f(x)\right). \end{aligned} \tag{4.19}$$

As we have

$$e(0,x) = \varphi_\theta^f(0,x) - x, \tag{4.20}$$

by denoting $\tilde{L}_f$ the maximum w.r.t. t of Lipschitz constants of f w.r.t. its second variable, we get the following estimate:

$$|e(t,x)| \leqslant \delta_{IC} + t\delta_{ODE} + \tilde{L}_f \int_0^t |e(s,x)|\,\mathrm{d}s. \tag{4.21}$$

Eventually, Grönwall lemma gives the estimate (4.11).

4.B Appendix: Implementations

4.B.1 Finite difference

In PINN's, derivatives w.r.t. t of φ_θ^f are used. In order to get an efficient approximation of the derivative $\frac{\partial \varphi_\theta^f}{\partial t}(t,x)$, we use a centered finite difference approximation, which is of order 2:

$$\frac{\varphi_\theta^f(t+\eta) - \varphi_\theta^f(t-\eta)}{2\eta} = \frac{\partial \varphi_\theta^f}{\partial t}(t,x) + \mathcal{O}(\eta^2). \tag{4.22}$$

In our simulations, we take $\eta = 10^{-5}$.

4.B.2 Choice of the parameters for numerical experiments

4.B.2.1 Hénon-Heiles system

Parameters	
# Math Parameters:	
Dynamical system:	Hénon-Heiles
Time for ODE simulation:	$T = 1$
Time step for ODE simulation:	$h = 0.01$
High oscillation parameter:	$\varepsilon \in \left\{10^{-5}, 0.001, 0.01, 0.1, 1\right\}$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^4$
Number of data:	$K = 1\,000\,000$
Proportion of data for training:	80% - $K_0 = 800\,000$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-4}$
Weight decay:	10^{-9}
Batch size (mini-batching for training):	200
Weight in discretization of E_{L^2}:	10^{-1}
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training:

⋆ $\varepsilon = 1$: 4 h 21 min 05 s

⋆ $\varepsilon = 0.1$: 3 h 53 min 55 s

⋆ $\varepsilon = 0.01$: ± 5 h

⋆ $\varepsilon = 0.001$: 4 h 00 min 14 s

⋆ $\varepsilon = 10^{-5}$: 4 h 23 min 42 s

4.B.2.2 Van der Pol oscillator

Parameters	
# Math Parameters:	
Dynamical system:	Van der Pol
Time for ODE simulation:	$T = 1$
Time step for ODE simulation:	$h = 0.01$ (for $\varepsilon \in \{0.1\}$ - $h = 0.001$ (for $\varepsilon \in \left\{0.001, 10^{-5}\right\}$)
High oscillation parameter:	$\varepsilon \in \left\{0.1, 10^{-5}\right\}$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 100\,000$
Proportion of data for training:	80% - $K_0 = 80\,000$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-4}$
Weight decay:	10^{-9}
Batch size (mini-batching for training):	200
Weight in discretization of E_{L^2}:	10^{-1}
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training:

⋆ $\varepsilon = 0.1$: 24 min 29 s

⋆ $\varepsilon = 0.001$: 1 h 05 min 46 s

⋆ $\varepsilon = 10^{-5}$: 21 min 48 s

BIBLIOGRAPHY

[1] George Anastassiou, *Quantitative approximations*, Chapman and Hall/CRC, 2000.

[2] George A Anastassiou, « General sigmoid based Banach space valued neural network approximation », *in*: *J. Computational Analysis and Applications* 31.*4* (2023), pp. 520–534.

[3] Clarissa Astuto, Mohammed Lemou, and Giovanni Russo, « Time multiscale modeling of sorption kinetics I: uniformly accurate schemes for highly oscillatory advection-diffusion equation », *in*: *arXiv preprint arXiv:2307.14001* (2023).

[4] Francis Bach, *Learning Theory from First Principles*, 2021.

[5] Dor Bank, Noam Koenigstein, and Raja Giryes, « Autoencoders », *in*: *Machine learning for data science handbook: data mining and knowledge discovery handbook* (2023), pp. 353–374.

[6] Weizhu Bao and Xiaofei Zhao, « Comparison of numerical methods for the nonlinear Klein-Gordon equation in the nonrelativistic limit regime », *in*: *Journal of Computational Physics* 398 (2019), p. 108886.

[7] Gal Berkooz, Philip Holmes, and John L Lumley, « The proper orthogonal decomposition in the analysis of turbulent flows », *in*: *Annual review of fluid mechanics* 25.*1* (1993), pp. 539–575.

[8] Wolf-Jürgen Beyn, Luca Dieci, Nicola Guglielmi, Ernst Hairer, Jesús María Sanz-Serna, and Marino Zennaro, « Current Challenges in Stability Issues for Numerical Differential Equations », *in*: *Cetraro: Springer* (2011).

[9] Nikolai Nikolaevich Bogoliubov and Iurii Alekseevich Mitropolśkii, *Asymptotic methods in the theory of non-linear oscillations*, vol. 10, CRC Press, 1961.

[10] Maxime Bouchereau, Philippe Chartier, Mohammed Lemou, and Florian Méhats, « Machine Learning Methods for Autonomous Ordinary Differential Equations », *in*: *arXiv preprint arXiv:2304.09036* (2023).

[11] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz, « Discovering governing equations from data by sparse identification of nonlinear dynamical systems », *in*: *Proceedings of the national academy of sciences* 113.*15* (2016), pp. 3932–3937.

[12] John Charles Butcher, *Numerical methods for ordinary differential equations*, John Wiley & Sons, 2016.

[13] Mari Paz Calvo, Philippe Chartier, Ander Murua, and Jesús María Sanz-Serna, « A stroboscopic numerical method for highly oscillatory problems », *in*: *Numerical analysis of multiscale computations: proceedings of a winter workshop at the Banff International Research Station 2009*, Springer, 2012, pp. 71–85.

[14] J.E. Campagne, *Les réseaux de neurones multi-couches: le comment et le pourquoi*, Notes et commentaires au sujet des conférences de S. Mallat du Collège de France, 2019.

[15] Fernando Casas and Vicente Martínez, *Advances in Differential Equations and Applications*, Springer, 2014.

[16] François Castella, Philippe Chartier, and Erwan Faou, « An averaging technique for highly oscillatory Hamiltonian problems », *in*: *SIAM Journal on Numerical Analysis* 47.*4* (2009), pp. 2808–2837.

[17] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert, « Multi-level Residual Networks from Dynamical Systems View », *in*: *International Conference on Learning Representations*, 2018.

[18] Philippe Chartier, *Introduction to the theory of Ordinary Differential Equations (in french)*, 2017, URL: `http://www.irisa.fr/ipso/perso/chartier/teaching.html`.

[19] Philippe Chartier, Nicolas Crouseilles, Mohammed Lemou, and Florian Méhats, « Averaging of highly-oscillatory transport equations », *in*: *Kinetic and Related Models* 13.*6* (2020), pp. 1107–1133.

[20] Philippe Chartier, Ernst Hairer, and Gilles Vilmart, « Numerical integrators based on modified differential equations », *in*: *Mathematics of computation* 76.*260* (2007), pp. 1941–1953.

[21] Philippe Chartier, Mohammed Lemou, Florian Méhats, and Gilles Vilmart, « A new class of uniformly accurate numerical schemes for highly oscillatory evolution equations », *in*: *Foundations of Computational Mathematics* 20 (2020), pp. 1–33.

[22] Philippe Chartier, Mohammed Lemou, Florian Méhats, and Xiaofei Zhao, « Derivative-free high-order uniformly accurate schemes for highly oscillatory systems », *in*: *IMA Journal of Numerical Analysis* 42.*2* (2022), pp. 1623–1644.

[23] Philippe Chartier, Joseba Makazaga, Ander Murua, and Gilles Vilmart, « Multi-revolution composition methods for highly oscillatory differential equations », *in*: *Numerische Mathematik* 128 (2014), pp. 167–192.

[24] Philippe Chartier, Ander Murua, and Jesús María Sanz-Serna, « A formal series approach to averaging: exponentially small error estimates », *in*: *Discrete and Continuous Dynamical Systems-Series A* 32.*9* (2012).

[25] Philippe Chartier, Ander Murua, and Jesús María Sanz-Serna, « Higher-order averaging, formal series and numerical integration I: B-series », *in*: *Foundations of Computational Mathematics* 10.*6* (2010), pp. 695–727.

[26] Philippe Chartier, Ander Murua, and Jesús María Sanz-Serna, « Higher-order averaging, formal series and numerical integration III: error bounds », *in*: *Foundations of Computational Mathematics* 15 (2015), pp. 591–612.

[27] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud, « Neural ordinary differential equations », *in*: *Advances in neural information processing systems* 31 (2018).

[28] Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong, « On the convergence of a class of Adam-type algorithms for non-convex optimization », *in*: *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[29] Junwoo Cho, Seungtae Nam, Hyunmo Yang, Seok-Bae Yun, Youngjoon Hong, and Eunbyung Park, « Separable physics-informed neural networks », *in*: *Advances in Neural Information Processing Systems* 36 (2024).

[30] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho, « Lagrangian Neural Networks », *in*: *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[31] George Cybenko, « Approximation by superpositions of a sigmoidal function », *in*: *Mathematics of control, signals and systems* 2.*4* (1989), pp. 303–314.

[32] Marco David and Florian Méhats, « Symplectic learning for Hamiltonian neural networks », *in*: *Journal of Computational Physics* 494 (2023), p. 112495.

[33] Tim De Ryck, Samuel Lanthaler, and Siddhartha Mishra, « On the approximation of functions by tanh neural networks », *in*: *Neural Networks* 143 (2021), pp. 732–750.

[34] Bruno Després, *Neural Networks and Numerical Analysis*, vol. 6, Walter de Gruyter GmbH & Co KG, 2022.

[35] Ronald A DeVore and Robert C Sharpley, « Besov spaces on domains », *in*: *Transactions of the American Mathematical Society* 335.*2* (1993), pp. 843–864.

[36] Qiang Du, Yiqi Gu, Haizhao Yang, and Chao Zhou, « The discovery of dynamics via linear multistep methods and deep learning: error estimation », *in*: *SIAM Journal on Numerical Analysis* 60.*4* (2022), pp. 2014–2045.

[37] Kang Feng, « Formal power series and numerical algorithms for dynamical systems », *in*: *Proceedings of international conference on scientific computation, Hangzhou, China, Eds. Tony Chan & Zhong-Ci Shi, Series on Appl. Math*, vol. 1, 1991, pp. 28–35.

[38] Bosco Garcia-Archilla, Jesús María Sanz-Serna, and Robert D Skeel, « Long-time-step methods for oscillatory differential equations », *in*: *SIAM Journal on Scientific Computing* 20.*3* (1998), pp. 930–963.

[39] Pierre Gillot, Akka Zemmari, Jenny Benois-Pineau, and Yurii Nesterov, *Algorithmes de Descente de Gradient Stochastique avec le filtrage des paramètres pour l'entraînement des réseaux à convolution profonds.*

[40] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse, « The reversible residual network: Backpropagation without storing activations », *in*: *Advances in neural information processing systems* 30 (2017).

[41] Pedro González de Alaiza Martínez and María Elena Vázquez-Cendón, « Operator-Splitting on Hyperbolic Balance Laws », *in*: *Advances in Differential Equations and Applications* (2014), pp. 279–287.

[42] William Gould, Jeffrey Pitblado, and William Sribney, *Maximum likelihood estimation with Stata*, Stata press, 2006.

[43] Pawan Goyal and Peter Benner, « Neural ordinary differential equations with irregular and noisy data », *in*: *Royal Society Open Science* 10.*7* (2023), p. 221475.

[44] Samuel Greydanus, Misko Dzamba, and Jason Yosinski, « Hamiltonian neural networks », *in*: *Advances in neural information processing systems* 32 (2019).

[45] Rémi Gribonval, Gitta Kutyniok, Morten Nielsen, and Felix Voigtlaender, « Approximation spaces of deep neural networks », *in*: *Constructive Approximation* 55.*1* (2022), pp. 259–367.

[46] Eldad Haber and Lars Ruthotto, « Stable architectures for deep neural networks », *in*: *Inverse problems* 34.*1* (2017), p. 014004.

[47] Eldad Haber, Lars Ruthotto, Elliot Holtham, and Seong-Hwan Jun, « Learning Across Scales—Multiscale Methods for Convolution Neural Networks », *in*: *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[48] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I*, Second, Springer Berlin Heidelberg, 1993, ISBN: 978-3-540-56670-0.

[49] Ernst Hairer and Christian Lubich, « Long-time energy conservation of numerical methods for oscillatory differential equations », *in*: *SIAM journal on numerical analysis* 38.*2* (2000), pp. 414–441.

[50] Ernst Hairer, Christian Lubich, and Gerhard Wanner, *Geometric Numerical Integration*, Mar. 2010, ISBN: 9783642051579.

[51] Ernst Hairer and Gilles Vilmart, « Preprocessed discrete Moser–Veselov algorithm for the full dynamics of a rigid body », *in*: *Journal of Physics A: Mathematical and General* 39.*42* (2006), p. 13225.

[52] Katsiaryna Haitsiukevich and Alexander Ilin, « Learning trajectories of hamiltonian systems with neural networks », *in*: (2022), pp. 562–573.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, « Identity mappings in deep residual networks », *in*: *European conference on computer vision*, Springer, 2016, pp. 630–645.

[54] John M Holte, « Discrete Gronwall lemma and applications », *in*: *MAA-NCS meeting at the University of North Dakota*, vol. 24, 2009, pp. 1–7.

[55] Yihao Hu, Tong Zhao, Shixin Xu, Lizhen Lin, and Zhiliang Xu, « NeuralPDE: a RNN based neural network for solving time dependent PDEs », *in*: *Communications in Information and Systems* 22.*2* (2022), pp. 223–245.

[56] A Ivanov, A Golovkina, and U Iben, « Polynomial neural networks and taylor maps for dynamical systems simulation and learning », *in*: *Frontiers in Artificial Intelligence and Applications* 325 (2020), pp. 1230–1237.

[57] Herbert Jaeger, « Echo state network », *in*: *scholarpedia* 2.*9* (2007), p. 2330.

[58] Pengzhan Jin, Zhen Zhang, Ioannis G Kevrekidis, and George Em Karniadakis, « Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks », *in*: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[59] Pengzhan Jin, Zhen Zhang, Aiqing Zhu, Yifa Tang, and George Em Karniadakis, « SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems », *in*: *Neural Networks* 132 (2020), pp. 166–179.

[60] Shi Jin, Zheng Ma, and Keke Wu, « Asymptotic-preserving neural networks for multiscale time-dependent linear transport equations », *in*: *Journal of Scientific Computing* 94.*3* (2023), p. 57.

[61] Patrick Kidger and Terry Lyons, « Universal approximation with deep narrow networks », *in*: (2020), pp. 2306–2327.

[62] Juš Kocijan, Agathe Girard, Blaž Banko, and Roderick Murray-Smith, « Dynamic systems identification with Gaussian processes », *in*: *Mathematical and Computer Modelling of Dynamical Systems* 11.*4* (2005), pp. 411–424.

[63] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar, « Neural operator: Learning maps between function spaces with applications to pdes », *in*: *Journal of Machine Learning Research* 24.*89* (2023), pp. 1–97.

[64] Isaac Elias Lagaris, Aristidis Likas, and Dimitrios I Fotiadis, « Artificial Neural Networks for Solving Ordinary and Partial Differential Equations », *in*: *IEEE TRANSACTIONS ON NEURAL NETWORKS* 9.*5* (1998), p. 987.

[65] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich, « FractalNet: Ultra-Deep Neural Networks without Residuals », *in*: *International Conference on Learning Representations*, 2016.

[66] Benedict Leimkuhler and Sebastian Reich, *Simulating hamiltonian dynamics*, 14, Cambridge university press, 2004.

[67] Wing Tat Leung, Guang Lin, and Zecheng Zhang, « NH-PINN: Neural homogenization-based physics-informed neural network for multiscale problems », *in*: *Journal of Computational Physics* 470 (2022), p. 111539.

[68] Jianfei Li, Han Feng, and Ding-Xuan Zhou, « SignReLU neural network and its approximation ability », *in*: *Journal of Computational and Applied Mathematics* 440 (2024), p. 115551.

[69] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao, « Independently recurrent neural network (indrnn): Building a longer and deeper rnn », *in*: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5457–5466.

[70] Zhengyi Li, Yanli Wang, Hongsheng Liu, Zidong Wang, and Bin Dong, « Solving the Boltzmann Equation with a Neural Sparse Representation », *in*: *SIAM Journal on Scientific Computing* 46.*2* (2024), pp. C186–C215.

[71] Xia Liu, « Approximating smooth and sparse functions by deep neural networks: optimal approximation rates and saturation », *in*: *Journal of Complexity* 79 (2023), p. 101783.

[72] Jianfeng Lu, Zuowei Shen, Haizhao Yang, and Shijun Zhang, « Deep network approximation for smooth functions », *in*: *SIAM Journal on Mathematical Analysis* 53.*5* (2021), pp. 5465–5506.

[73] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong, « Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations », *in*: (2018), pp. 3276–3285.

[74] Yulong Lu, Li Wang, and Wuzhe Xu, « Solving multiscale steady radiative transfer equation using neural networks with uniform stability », *in*: *Research in the Mathematical Sciences* 9.*3* (2022), p. 45.

[75] Marios Mattheakis, David Sondak, Akshunna S Dogra, and Pavlos Protopapas, « Hamiltonian neural networks for solving equations of motion », *in*: *Physical Review E* 105.*6* (2022), p. 065305.

[76] Micheal L Mavrovouniotis and S Chang, « Hierarchical neural networks », *in*: *Computers & chemical engineering* 16.*4* (1992), pp. 347–369.

[77] Larry Medsker and Lakhmi C Jain, *Recurrent neural networks: design and applications*, CRC press, 1999.

[78] Leon Migus, Yuan Yin, Jocelyn Ahmed Mazari, and Patrick Gallinari, « Multi-scale physical representations for approximating pde solutions with graph neural operators », *in*: *Topological, Algebraic and Geometric Learning Workshops 2022*, PMLR, 2022, pp. 332–340.

[79] Léon Migus, Julien Salomon, and Patrick Gallinari, « Stability of implicit neural networks for long-term forecasting in dynamical systems », *in*: *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.

[80] Etienne Le Naour, Louis Serrano, Léon Migus, Yuan Yin, Ghislain Agoua, Nicolas Baskiotis, Patrick Gallinari, and Vincent Guigue, « Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations », *in*: *arXiv preprint arXiv:2306.05880* (2023).

[81] Duong Nguyen, Said Ouala, Lucas Drumetz, and Ronan Fablet, « Em-like learning chaotic dynamics from noisy and partial observations », *in*: *arXiv preprint arXiv:1903.10335* (2019).

[82] Håkon Noren, « Learning Hamiltonian Systems with Mono-Implicit Runge-Kutta Methods », *in*: *International Conference on Geometric Science of Information*, Springer, 2023, pp. 552–559.

[83] Håkon Noren, Sølve Eidnes, and Elena Celledoni, « Learning dynamical systems from noisy data with inverse-explicit integrators », *in*: *arXiv preprint arXiv:2306.03548* (2023).

[84] Keiron O'shea and Ryan Nash, « An introduction to convolutional neural networks », *in*: *arXiv preprint arXiv:1511.08458* (2015).

[85] Christian Offen and Sina Ober-Blöbaum, « Symplectic integration of learned Hamiltonian systems », *in*: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 32.*1* (2022), p. 013122.

[86] Christian W Omlin and C Lee Giles, « Constructing deterministic finite-state automata in recurrent neural networks », *in*: *Journal of the ACM (JACM)* 43.*6* (1996), pp. 937–972.

[87] Lawrence M Perko, « Higher order averaging and related methods for perturbed periodic and quasi-periodic systems », *in*: *SIAM Journal on Applied Mathematics* 17.*4* (1969), pp. 698–724.

[88] Allan Pinkus, « Approximation theory of the MLP model in neural networks », *in*: *Acta numerica* 8 (1999), pp. 143–195.

[89] Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park, « Hypersolvers: Toward fast continuous-depth models », *in*: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21105–21117.

[90] Peter J Prince and John R Dormand, « High order embedded Runge-Kutta formulae », *in*: *Journal of computational and applied mathematics* 7.*1* (1981), pp. 67–75.

[91] Tim Prokosch, « Solving differential equations by means of Physics Informed Neural Networks », *in*: (2022).

[92] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis, « Machine learning of linear differential equations using Gaussian processes », *in*: *Journal of Computational Physics* 348 (2017), pp. 683–693.

[93] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis, « Multistep neural networks for data-driven discovery of nonlinear dynamical systems », *in*: *arXiv preprint arXiv:1801.01236* (2018).

[94] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis, « Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations », *in*: *arXiv preprint arXiv:1711.10561* (2017).

[95] Francesco Regazzoni, Luca Dedè, and Alfio Quarteroni, « Machine learning for fast and reliable solution of time-dependent differential equations », *in*: *Journal of Computational physics* 397 (2019), p. 108852.

[96] Ramiro Rico-Martinez and Ioannis G Kevrekidis, « Continuous time modeling of nonlinear systems: A neural network-based approach », *in*: *IEEE International Conference on Neural Networks*, IEEE, 1993, pp. 1522–1525.

[97] Jan A Sanders, Ferdinand Verhulst, and James Murdock, *Averaging methods in nonlinear dynamical systems*, vol. 59, Springer, 2007.

[98] Jesús María Sanz-Serna, « Mollified impulse methods for highly oscillatory differential equations », *in*: *SIAM journal on numerical analysis* 46.*2* (2008), pp. 1040–1059.

[99] Jürgen Schmidhuber, « Learning complex, extended sequences using the principle of history compression », *in*: *Neural Computation* 4.*2* (1992), pp. 234–242.

[100] Mike Schuster and Kuldip K Paliwal, « Bidirectional recurrent neural networks », *in*: *IEEE transactions on Signal Processing* 45.*11* (1997), pp. 2673–2681.

[101] Louis Serrano, Leon Migus, Yuan Yin, Jocelyn Ahmed Mazari, and Patrick Gallinari, « Infinity: Neural field modeling for reynolds-averaged navier-stokes equations », *in*: *arXiv preprint arXiv:2307.13538* (2023).

[102] Eli Stevens, Luca Antiga, and Thomas Viehmann, *Deep learning with PyTorch*, Manning Publications, 2020.

[103] Siping Tang, Xinlong Feng, Wei Wu, and Hui Xu, « Physics-informed neural networks combined with polynomial interpolation to solve nonlinear partial differential equations », *in*: *Computers & Mathematics with Applications* 132 (2023), pp. 48–62.

[104] Dmitriy Tarkhov, Tatiana Lazovskaya, and Galina Malykhina, « Constructing physics-informed neural networks with architecture based on analytical modification of numerical methods by solving the problem of modelling processes in a chemical reactor », *in*: *Sensors* 23.*2* (2023), p. 663.

[105] Kurt Tutschku, « Recurrent multilayer perceptrons for identification and control: The road to applications », *in*: *Univ. Würzburg, Germany, ser. Research Report Series* (1995), p. 7.

[106] Bin Wang and Yaolin Jiang, « Structure-preserving algorithms with uniform error bound and long-time energy conservation for highly oscillatory Hamiltonian systems », *in*: *Journal of Scientific Computing* 95.*3* (2023), p. 66.

[107] Jing Wang, Zheng Li, Pengyu Lai, Rui Wang, Di Yang, and Hui Xu, « Multiscale Modelling with Physics-informed Neural Network: from Large-scale Dynamics to Small-scale Predictions in Complex Systems », *in*: *arXiv preprint arXiv:2402.05067* (2024).

[108] Yu Wang, Yuxuan Yin, Karthik Somayaji NS, Ján Drgoňa, Malachi Schram, Mahantesh Halappanavar, Frank Liu, and Peng Li, « Semi-supervised Learning of Dynamical Systems with Neural Ordinary Differential Equations: A Teacher-Student Model Approach », *in*: 38.*14* (2024), pp. 15698–15705.

[109] Yuting Weng and Dezhi Zhou, « Multiscale physics-informed neural networks for stiff chemical kinetics », *in*: *The Journal of Physical Chemistry A* 126.*45* (2022), pp. 8534–8543.

[110] Alistair White, Niki Kilbertus, Maximilian Gelbrecht, and Niklas Boers, « Stabilized Neural Differential Equations for Learning Dynamics with Explicit Constraints », *in*: *Advances in Neural Information Processing Systems* 36 (2024).

[111] Wikipedia contributors, *Gated recurrent unit — Wikipedia, The Free Encyclopedia*, 2022, URL: `https://en.wikipedia.org/wiki/Gated_recurrent_unit`.

[112] Wikipedia contributors, *Long short-term memory — Wikipedia, The Free Encyclopedia*, 2022, URL: `https://en.wikipedia.org/wiki/Long_short-term_memory`.

[113] Wikipedia contributors, *Recurrent neural network — Wikipedia, The Free Encyclopedia*, 2022, URL: `https://en.wikipedia.org/wiki/Recurrent_neural_network`.

[114] Keke Wu, Xiong-Bin Yan, Shi Jin, and Zheng Ma, « Capturing the diffusive behavior of the multiscale linear transport equations by Asymptotic-Preserving Convolutional DeepONets », *in*: *Computer Methods in Applied Mechanics and Engineering* 418 (2024), p. 116531.

[115] Zixue Xiang, Wei Peng, Weien Zhou, and Wen Yao, « Hybrid finite difference with the physics-informed neural network for solving PDE in complex geometries », *in*: *arXiv preprint arXiv:220-2.07926* (2022).

[116] Yuichi Yamashita and Jun Tani, « Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment », *in*: *PLoS computational biology* 4.*11* (2008), e1000220.

[117] Aiqing Zhu, Pengzhan Jin, Beibei Zhu, and Yifa Tang, « Inverse modified differential equations for discovery of dynamics », *in*: *arXiv preprint arXiv:2009.01058* (2020).

[118] Aiqing Zhu, Pengzhan Jin, Beibei Zhu, and Yifa Tang, « On numerical integration in neural ordinary differential equations », *in*: *International Conference on Machine Learning*, PMLR, 2022, pp. 27527–27547.

[119] Aiqing Zhu, Beibei Zhu, Jiawei Zhang, Yifa Tang, and Jian Liu, « VPNets: Volume-preserving neural networks for learning source-free dynamics », *in*: *Journal of Computational and Applied Mathematics* 416 (2022), p. 114523.

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Modélisation de phénomènes hautement oscillants par réseaux de neurones

Mot clés : Equations différentielles autonomes et fortement oscillantes, analyse rétrograde, méthodes numériques uniformément précises, réseaux de neurones, erreur numérique

Résumé : Cette thèse porte sur l'application du Machine Learning à l'étude d'équations différentielles fortement oscillantes. Plus précisément, on s'intéresse à une manière d'approcher de manière précise et avec le moins de calculs possible la solution d'une équation différentielle en s'aidant de réseaux de neurones. Tout d'abord, le cas autonome est étudié, où les propriétés de l'analyse rétrograde et des réseaux de neurones sont utilisés afin d'améliorer des méthodes numériques existantes, puis une généralisation au cas fortement oscillant est proposée afin d'améliorer un schéma numérique d'ordre un spécifique à ce cas de figure. Ensuite, les réseaux de neurones sont utilisés afin de remplacer les calculs préalables nécessaires à l'implémentation de méthodes numériques uniformément précises permettant d'approcher les solutions d'équations fortement oscillantes, que ce soit en partant des travaux mis en oeuvre pour le cas autonome, ou bien en utilisant une structure de réseau de neurone intégrant directement la structure de l'équation.

Title: Modelling of highly oscillatory phenomenon by neural networks

Keywords: Autonomous and highly oscillatory differential equations, backward analysis, uniformly accurate numerical methods, neural networks, numerical error

Abstract: This thesis focuses on the application of Machine Learning to the study of highly oscillatory differential equations. More precisely, we are interested in an approach to accurately approximate the solution of a differential equation with the least amount of computations, using neural networks. First, the autonomous case is studied, where the properties of backward analysis and neural networks are used to enhance existing numerical methods. Then, a generalization to the strongly oscillating case is proposed to improve a specific first-order numerical scheme tailored to this scenario. Subsequently, neural networks are employed to replace the necessary precomputations for implementing uniformly accurate numerical methods to approximate solutions of strongly oscillating equations. This can be done either by building upon the work done for the autonomous case or by using a neural network structure that directly incorporates the equation's structure.