Formalisations d'analyses d'erreurs en analyse numérique et en arithmétique à virgule flottante

Florian Faissole

université PARIS-SACLAY

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

NNT : 2019SACLS594

Thèse de doctorat

Formalisations d'analyses d'erreurs en analyse numérique et en arithmétique à virgule flottante

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université Paris-Sud

École doctorale n°580 Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Orsay, le 13/12/2019, par

FLORIAN FAISSOLE

Composition du Jury :

Florent Hivert Professeur, Université Paris-Sud	Président
Yves Bertot Directeur de recherche, Inria Sophia Antipolis	Rapporteur
Paul Zimmermann Directeur de recherche, Inria Nancy Grand Est	Rapporteur
Stef Graillat Professeur, Sorbonne Université	Examinateur
Assia Mahboubi Chargée de recherche, Inria, LS2N	Examinatrice
Sylvie Boldo Directrice de recherche, Inria Saclay Île-de-France	Directrice de thèse
Alexandre Chapoutot Maître de conférences, ENSTA Paris	Co-directeur de thèse

Remerciements

La thèse est avant tout une rencontre : une rencontre avec la science mais également une rencontre avec ceux qui la font avancer et la transmette. Cela fait près de quatre ans que Sylvie Boldo a accepté de diriger mes travaux de recherche puis ma thèse de doctorat. Je lui suis très reconnaissant pour la qualité remarquable de son encadrement, tant sur le plan scientifique qu'humain. Sylvie m'a transmis le goût de l'exigence et de la rigueur tout en m'accordant toute sa confiance et la dose idéale d'indépendance. Elle a constamment fait preuve d'empathie et m'a apporté son aide pour toutes sortes de démarches. Je remercie également mon co-encadrant Alexandre Chapoutot qui m'a apporté une myriade de nouvelles connaissances et m'a permis d'élargir l'état de la littérature dressé dans ce manuscrit. Le résultat n'aurait sans doute pas été le même sans sa bienveillance et ses conseils.

Je tiens à remercier Yves Bertot et Paul Zimmermann d'avoir accepté d'être rapporteurs de ma thèse. Je suis heureux d'avoir pu bénéficier de leurs conseils scientifiques qui m'ont permis d'améliorer la qualité du document. J'ai par ailleurs eu le plaisir de les rencontrer par le passé lors d'événements au cours desquels nous avons pu échangé sur divers sujets scientifiques. J'adresse également mes remerciements aux autres personnes qui ont accepté d'être membres de mon jury, à savoir Stef Graillat, Florent Hivert et Assia Mahboubi.

J'ai eu la chance de pouvoir préparer mon doctorat dans un environnement très enrichissant, entouré de personnes passionnantes et passionnés. Je remercie à ce titre l'ensemble des personnes de l'équipe VALS/Toccata qui ont croisé mon chemin : Thibault Balabonski, Lucas Baudin, Benedikt Becker, Véronique Benzaken, Valentin Blot, Arthur Charguéraud, Martin Clochard, Sylvain Conchon, Évelyne Contejean, Albin Coquereau, Sylvain Dailler, David Declerck, Stefania Dumbrava, Benjamin Farinier, Jean-Christophe Filliâtre, Clément Fumex, Diane Gallois-Wong, Quentin Garchery, Marie-Claude Gaudel, Léon Gondelman, Mohammed Hachmaoui, Thibault Hilaire, Jacques-Henri Jourdan, Chantal Keller, Delphine Longuet, Julien Lopez, Cláudio Lourenço, Claude Marché, Guillaume Melquiond, Kim Nguyen, Hai Nguyen-Van, Georges Ouffoue, Andrei Paskevich, Christine Paulin, Robin Pelle, Mário Pereira, Raphaël Rieu-Helft, Mattias Roux, Florian Steinberg, Frédéric Tuong, Xavier Urbain, Frédéric Voisin, Burkhart Wolff et Fatiha Zaidi. En particulier, merci à Christine Paulin pour son aide au cours de mes premiers pas avec Coq, à Guillaume Melquiond qui répond toujours très précisément à mes questions, à Kim Nguyen, Sylvain Conchon et Frédéric Voisin qui ont essayé de m'aider à affronter les difficultés administratives même les plus complexes et à Marie-Claude Gaudel pour nos discussions sur l'art et les belles lettres ainsi que pour son engagement auprès de la *Fondation Paris-Sud*. Je remercie par ailleurs l'ensemble des membres du LRI et en particulier les personnes avec lesquelles j'ai collaboré au cours de ma mission d'enseignement à l'Université Paris-Sud. Enfin, je dois la qualité de mes conditions de travail à l'aide précieuse de Katia Évrat, mais aussi de

Régine Bricquet et de l'ensemble du personnel administratif d'Inria. Merci également à Isabelle Huteau pour nos nombreuses discussions et son intérêt pour mes thématiques de recherche.

Les travaux présentés n'auraient pas vu le jour sans mes co-auteurs. Je remercie à ce titre, en plus de mes encadrants de thèse, les membres du projet ELFIC/MILC, notamment François Clément, Vincent Martin et Micaela Mayero. Je remercie également Vincent Tourneur pour le travail réalisé ensemble au cours de son stage de M1.

Au cours de ma thèse, j'ai par deux fois collaboré avec des chercheurs étrangers sur des sujets variés. *A special thanks to Bas Spitters who hosted me at Aarhus University. I would like thank Lars Birkedal and Léo Stefanesco too for this very nice danish experience. I would like thank George Constantinides, David Thomas and all the PhD students and postdoctoral researchers who hosted me at Imperial College London. A special thank to Wiesia Hsissen because she helped me to better understand the UK administrative processes.*

Depuis le 1er octobre 2019, je suis chercheur permanent en méthodes formelles au centre de Recherche de Mitsubishi Electric à Rennes. Je remercie les membres de mon équipe, en particulier David Mentré, Denis Cousineau et Benoît Boyer pour leur accueil. Je remercie mes responsables, parmi lesquels David Mottier, Magali Branchereau, Loïc Brunel et David Mentré, qui m'ont accordé leur confiance et m'ont laissé terminer ma thèse de doctorat dans des conditions plus qu'optimales et confortables.

Je tiens à remercier les entités qui ont financé mes recherches, notamment le *Labex Digicosme* de l'Université Paris-Saclay qui a financé mon contrat doctoral.

Je n'oublie pas l'ensemble des personnes qui m'ont mis le pied à l'étrier et m'ont fait aimé l'informatique et les mathématiques, en particulier Françoise Massy, Thomas Rey, Catherine Dubois, Xavier Urbain, Karim Berkani, Mélissa Issad mais aussi mon amie Amélie Delga qui m'a encouragé à suivre ce parcours.

J'en profite pour remercier mes ami(e)s, qui m'ont soutenu au cours des différentes périodes de ma thèse. De peur d'oublier des personnes, je ne citerai aucun nom, mais les personnes concernées se renconnaitront. Je remercie également toute ma famille ainsi que ma belle-famille pour leur indéfectible soutien (merci à Yveline et Jean-Louis de me supporter et de m'aider dans l'organisation du traditionnel *pot*).

Je remercie tout particulièrement mes parents Francine et Jean-Pierre car ils ont toujours cru en moi et ont tout fait pour que je sois heureux et épanoui dans ma vie professionnelle. Je leur dois en grande partie ce travail.

Enfin, je remercie énormément Sophie, qui m'a accompagné et soutenu à chaque étape de ma thèse avec la bienveillance et l'affection qui la caractérise.

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Vers la chasse aux bugs : Depuis les années 1980, l'augmentation de la puissance de calcul des ordinateurs a conduit à l'informatisation de systèmes de plus en plus complexes. De nos jours, la plupart des objets de notre quotidien (voitures, trains, systèmes de transports d'électricité, etc) reposent sur des programmes informatiques de taille conséquente. Malheureusement, la spécification du comportement attendu de ces systèmes est souvent négligée ou rédigée de manière informelle. De surcroît, par manque de temps, de moyens ou d'outils, la validation des systèmes informatiques est non-exhaustive et parfois trop légère, si bien que de nombreux bugs passent à travers les mailles de la phase de tests. Lorsqu'ils interviennent dans des applications critiques, ces bugs peuvent avoir des répercussions financières ou humaines qui sont catastrophiques. Nous pouvons par exemple évoquer le bug de la fusée Ariane 5 en 1996 [16], qui a explosé en plein vol parce que la valeur de l'accélération à stocker dépassait la valeur maximale de la structure de donnée prévue à cet effet. Évoquons également le dysfonctionnement de la machine de radiothérapie Therac-25 entre 1985 et 1987 [137], qui a causé la mort de cinq personnes après leur avoir administré une dose de radioactivité vingt fois trop élevée. Dans les deux cas, la négligence des développeurs a été mise en cause.

L'une des sources de bugs les plus fréquemment négligées est liée à la précision finie des structures de données utilisées dans les programmes informatiques, *e.g.* les nombres à virgule fixe ou flottante, et aux erreurs d'arrondi associées. En 1991, durant la guerre du Golfe, le système anti-missile Patriot MIM-104 [202] a raté un missile irakien qui causera la mort de vingt-huit soldats américains, avec un tir à plus de 600 mètres de la cible. L'une des données utilisées par le système Patriot était le temps (en secondes) écoulé depuis sa mise en route. Or, les calculs ont été implémentés en arithmétique à virgule fixe, avec une précision de 24 bits. Au moment où le bug s'est produit, le système était en route depuis plus de 20 heures et la valeur prise par le temps écoulé était si grande que d'importantes erreurs d'arrondi se sont produites dans les calculs. Ces erreurs ont conduit à un décalage de près d'une demi-seconde entre le temps calculé et le temps effectivement écoulé, entraînant le désastreux échec du tir balistique.

Assistant de preuves Coq : Ces catastrophes ont mené à prendre conscience de l'importance d'une vérification rigoureuse du logiciel, notamment lorsque le code source est utilisé par des applications critiques. Les méthodes formelles sont une famille de techniques permettant de modéliser des systèmes informatiques et des spécifications de manière rigoureuse puis de raisonner sur ces systèmes pour vérifier qu'ils satisfont leur spécification. Nous distinguons plusieurs classes de méthodes formelles, comme l'interprétation abstraite [53], le model-checking [8], la vérification déductive de programmes [76, 75] et la preuve interactive *via* un assistant de preuves [82]. Dans ce manuscrit, l'approche adoptée est la preuve interactive. Il existe un certain nombre d'assistants de preuves fondés sur différents formalismes logiques, *e.g.* Mizar [157], Isabelle/HOL [165], PVS [160], ou Coq [17] (dans le chapitre 3, nous présentons certaines formalisations dans ces assistants de preuves). Dans cette thèse, nous utilisons Coq.

Coq [17] est un assistant de preuves qui inclut un langage de spécification appelé Gallina, qui permet de spécifier des concepts mathématiques et des propriétés sur ces concepts. Un théorème Coq est spécifié à partir des concepts inhérents à la théorie considérée, puis démontré *via* des règles logiques ou d'autres résultats démontrés en amont. Les étapes consécutives de la preuve peuvent être menées en utilisant un langage de tactiques dédiées qui sont assemblées pour former un script de preuves. Coq vérifie ensuite le script de preuves complet par typage. Le langage de spécification de Coq repose sur une théorie des types d'ordre supérieur appelée calcul des constructions inductives (CIC) [166] qui inclut notamment des types dépendants et inductifs. La logique de Coq est intuitionniste, *i.e.* l'axiome du tiers-exclu n'est pas inclus par défaut. Le calcul des constructions inductives est fondé sur la correspondance de Curry-Howard ou correspondance « preuve-programme », qui fait correspondre les types à des propositions et les termes à des démonstrations. Ainsi, dans Coq, $x : t$ peut être interprété aussi bien comme « x est un élément de type t » que comme « x est une preuve de t ». Le mécanisme d'extraction de Coq [136] permet d'obtenir un programme OCaml ou Haskell correct par construction à partir d'une preuve Coq de sa spécification.

Analyse numérique des équations différentielles et implémentation en arithmétique à virgule flottante : La résolution numérique des équations différentielles constitue un domaine pouvant être sujet aux bugs et aux problèmes de précision numérique. La majorité des lois fondamentales rencontrées en physique, en chimie, en biologie ou en économie peuvent être modélisées à l'aide d'équations différentielles ordinaires ou aux dérivées partielles. Ces équations permettent par exemple de caractériser la décharge d'un condensateur [197], l'équation de la chaleur [94] ou la décroissance radioactive d'un élément chimique [185]. Malheureusement, la résolution directe de ces équations est rarement possible ou peu commode. C'est pourquoi des méthodes de résolution numérique ont été mises au point, permettant d'obtenir de manière efficace une solution approchée « suffisamment » précise.

Dans le cas des équations différentielles ordinaires, les phénomènes étudiés représentent généralement l'évolution temporelle d'une quantité physique. Les méthodes de

résolution associées reposent sur la discrétisation d'un intervalle de temps et le calcul itératif des valeurs approchées de la fonction inconnue aux différents points de la discrétisation. Les méthodes de Runge-Kutta [45, 59] sont parmi les plus simples et pourtant les plus utilisées en recherche fondamentale ou appliquée à l'industrie. Dans le cas des équations aux dérivées partielles, les phénomènes étudiés sont plus complexes et souvent liés au comportement d'un flux au contact d'une surface ou d'un volume, comme l'écoulement de l'air sur le profil d'un avion ou du sang dans une artère. La méthode des éléments finis [64] est l'une des méthodes de résolution les plus répandues. Elle consiste à approcher le volume sur lequel est définie la fonction à intégrer, *e.g.* l'artère ou le profil d'aile d'avion, par un maillage de polyèdres réguliers appelés éléments finis sur lequel est résolu un système d'équations plus simples.

Ces méthodes sont par nature sources d'approximations et sont associées à une erreur mathématique appelée erreur de méthode. Une propriété importante des méthodes numériques est la propriété de convergence. Pour les méthodes de Runge-Kutta, cela signifie que l'erreur de méthode converge vers 0 lorsque le pas d'intégration tend vers 0. Pour la méthode des éléments finis, cela signifie que l'erreur de méthode converge vers 0 lorsque la finesse du maillage est augmentée. Les numériciens s'intéressent également à d'autres propriétés numériques importantes comme la stabilité, une méthode étant considérée comme stable lorsqu'elle n'amplifie pas trop d'éventuelles erreurs initiales.

En général, les quantités numériques manipulées par les méthodes de résolution d'équations différentielles sont représentées en machine avec un nombre fini de bits, si bien que seul un nombre fini de valeurs réelles sont représentables. Les nombres à virgule flottante [155, 83] sont l'une des structures de données les plus fréquemment utilisées pour représenter ces quantités. Lorsque le résultat d'un calcul n'est pas représentable dans le format flottant considéré, il est arrondi vers l'un des nombres flottants qui l'encadrent, le choix de ce nombre dépendant du mode d'arrondi utilisé. Chaque étape de calcul d'un algorithme itératif peut donc potentiellement conduire à des erreurs d'arrondi, qui peuvent s'accumuler au fil des itérations.

Ainsi, aux erreurs de méthode associées aux schémas numériques se rajoutent les erreurs d'arrondi, qui constituent une source supplémentaire d'inexactitude. Trop souvent, les numériciens ne se préoccupent pas de ces erreurs, difficiles à estimer dans le cas général et souvent considérées comme négligeables par rapport aux erreurs de méthode. Néanmoins, en toute généralité, il n'y a pas d'argument rigoureux assurant que les erreurs d'arrondi ne s'amplifient pas au point de dominer les erreurs de méthode et de rendre le schéma numérique inutilisable. Cela est d'autant plus problématique que les méthodes plus précises d'un point de vue mathématique, *i.e.* dont l'erreur de méthode est faible, induisent généralement un nombre plus important d'opérations flottantes et peuvent entraîner des erreurs d'arrondi plus importantes. À cela peut s'ajouter l'occurrence de comportements exceptionnels comme les dépassements de capacité inférieurs (respectivement supérieurs), qui surviennent lorsque le résultat d'un calcul intermédiaire de l'algorithme est trop petit (respectivement trop grand) pour être stocké dans le format flottant considéré.

1.2 Contributions et plan

1.2.1 Prérequis et état de l'art

Le chapitre 2 introduit les concepts fondamentaux sur lesquels se basent nos contributions. Les notions introduites concernent l'arithmétique à virgule flottante, les bibliothèques Coq utilisées et la résolution numérique des équations différentielles ordinaires.

Le chapitre 3 constitue un état de l'art concernant l'analyse des erreurs d'arrondi dans l'implémentation de méthodes numériques et les formalisations d'analyse numérique et fonctionnelle dans les assistants de preuves.

1.2.2 Formalisation d'un algorithme en arrondi correct pour le calcul de moyenne de nombres flottants décimaux

La partie I du document, *i.e.* le chapitre 4, présente la formalisation d'un algorithme en arrondi correct pour le calcul de la moyenne de deux nombres flottants décimaux.

Il est dit d'un algorithme implémenté en précision finie qu'il satisfait la propriété d'arrondi correct lorsque le résultat qu'il renvoie est le résultat représentable le plus proche possible. La norme IEEE-754 [113] impose que les opérations élémentaires satisfassent la propriété d'arrondi correct et conseille qu'elle soit satisfaite par certaines fonctions élémentaires comme les fonctions trigonométriques. En revanche, la propriété est rarement satisfaite par des algorithmes plus élaborés, *e.g.* les programmes numériques.

Les formats flottants décimaux sont de nos jours de plus en plus répandus, notamment dans les applications bancaires. Malheureusement, les propriétés d'un programme implémenté en base 2 ne sont pas nécessairement transposables à la base 10.

Boldo [24] a par exemple proposé un algorithme calculant la moyenne de deux nombres flottants binaires et satisfaisant la propriété d'arrondi correct. Nous montrons que cet algorithme ne satisfait pas l'arrondi correct en base 10 et proposons un algorithme relativement efficace qui calcule l'arrondi correct de la moyenne de deux nombres flottants décimaux. Nous proposons par ailleurs une formalisation en Coq de l'algorithme et de sa preuve de correction.

Publications. *Ce travail a fait l'objet d'un article avec S. Boldo et V. Tourneur [31].*

1.2.3 Analyse formelle des erreurs d'arrondi de méthodes de Runge-Kutta

La seconde partie du document propose la mise au point et la formalisation en Coq d'une méthodologie générique pour borner les erreurs d'arrondi de méthodes de Runge-Kutta appliquées à des systèmes linéaires. Le chapitre 5 est voué aux systèmes linéaires scalaires, *i.e.* les équations différentielles de la forme $y' = \lambda y$ avec $y, \lambda \in \mathbb{R}$. Le chapitre 6 généralise les résultats du chapitre 5 aux systèmes linéaires matriciels, *i.e.* de la forme $y' = Ay$ avec $y \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n}$.

Méthodologie générique d'analyse des erreurs d'arrondi : Nous présentons un ensemble de résultats génériques qui servent d'outils pour construire les bornes d'erreurs. Ces résultats sont paramétrés par l'équation différentielle considérée, la méthode de résolution adoptée, le format flottant et la forme des implémentations, *i.e.* l'ordre d'évaluation des opérations flottantes.

Nous nous limitons aux systèmes linéaires, c'est-à-dire aux équations différentielles de la forme $y' = Ay$ avec y un vecteur de $\mathbb{R}^n$ et A une matrice carrée de taille n. Les résultats présentés utilisent les propriétés fines du format d'arithmétique à virgule flottante, certaines propriétés mathématiques des méthodes, *e.g.* la stabilité numérique, ainsi que les propriétés des opérations matricielles.

L'idée générale est de distinguer les erreurs dites *locales*, *i.e.* qui se produisent à une itération donnée du schéma, et *globales*, *i.e.* accumulées au cours des itérations. Nous proposons des résultats génériques qui, pour n'importe quelle implémentation, permettent de construire une borne sur les erreurs locales des méthodes de Runge-Kutta. Nous proposons ensuite un résultat permettant de construire une borne sur l'erreur globale à partir des bornes sur les erreurs locales des itérations précédentes.

Ces résultats ont été instanciés à des méthodes de Runge-Kutta classiques et très fréquemment utilisées, comme les méthodes d'Euler et de Runge-Kutta 2, *via* des applications successives et mécaniques des lemmes génériques.

Une particularité de ce travail est la gestion des comportements exceptionnels de l'arithmétique à virgule flottante, souvent négligés dans les travaux d'analyse d'erreurs d'arrondi. Nous tenons plus précisément compte des dépassements graduels de capacité inférieurs. Dans le cas linéaire scalaire, nous tenons également compte des dépassements de capacité supérieurs.

Formalisation complète de l'analyse d'erreurs : La démonstration des résultats sur les erreurs d'arrondi est délicate et fastidieuse. Afin d'augmenter le niveau de confiance accordé à notre méthodologie, nous avons formalisé l'ensemble de l'analyse d'erreurs dans l'assistant de preuves Coq. Cela suppose la cohabitation de raisonnements sur les nombres flottants et de raisonnements sur les matrices et les opérations matricielles. Nous combinons pour cela la bibliothèque Flocq d'arithmétique des ordinateurs [37, 38] ainsi que la spécification des vecteurs et matrices de la bibliothèque Mathematical Components [144].

Une partie importante de la formalisation concerne l'analyse des erreurs d'arrondi d'opérations matricielles, *e.g.* les produits matrice-vecteur et matrice-matrice. Cette partie de la formalisation inclut la définition des opérations matricielles arrondies et la formalisation de propriétés sur les normes de matrices. L'analyse des erreurs d'arrondi des produits matriciels est technique et repose sur les bornes d'erreurs d'arrondi des sommations de nombres flottants et du produit scalaire de vecteurs de nombres réels.

Nous avons ensuite formalisé les méthodes de Runge-Kutta et leurs implémentations dans Coq, ainsi que les notions d'erreurs d'arrondi locales et globales pour ces méthodes numériques. La démonstration formelle des lemmes génériques permettant de gérer les erreurs locales et globales est ensuite relativement proche de la preuve papier.

L'application des résultats génériques à des méthodes de Runge-Kutta classiques s'avère besogneuse, d'autant plus que nous souhaitons simplifier les bornes pour les rendre plus lisibles. Pour pallier ce problème, nous utilisons un ensemble de tactiques automatiques existantes permettant d'automatiser une partie du raisonnement.

Publications. *Ces travaux ont fait l'objet de deux articles avec S. Boldo et A. Chapoutot. Le premier article [29] présente une version non formalisée de ce travail pour le cas scalaire et ne tenant pas compte des comportements exceptionnels. Le second article [30] étend [29] en tenant compte des dépassements de capacité inférieurs et supérieurs. Concernant le passage au cas des systèmes matriciels et la formalisation des résultats, seule une version préliminaire du travail a pour l'heure été publiée [70].*

1.2.4 Formalisation de résultats d'analyse fonctionnelle

Les méthodes numériques de résolution des équations aux dérivées partielles, *e.g.* la méthode des éléments finis, reposent sur des fondements mathématiques bien plus conséquents que les équations différentielles ordinaires. La troisième partie de cette thèse présente une formalisation de ces fondements mathématiques fondée sur Coquelicot, une extension conservative de la bibliothèque standard des réels développée par Boldo, Lelay et Melquiond [35, 134]. Le chapitre 7 est dédié à la formalisation des espaces de Hilbert et de leurs propriétés, ainsi qu'à la formalisation des espaces de fonctions linéaires continues. Le chapitre 8 est voué à la formalisation de résultats fondamentaux d'analyse fonctionnelle comme le théorème de Lax–Milgram et la complétude des sous-espaces de dimension finie d'espaces de Hilbert.

Espaces de Hilbert et espaces fonctionnels : La méthode des éléments finis permet d'approcher la solution d'équations aux dérivées partielles qui modélisent des phénomènes physiques sur des espaces irréguliers, *e.g.* le flux du sang dans une artère. La structure algébrique permettant d'abstraire ces volumes irréguliers est la structure d'espace de Hilbert, *i.e.* un module complet muni d'un produit scalaire.

Nous formalisons la structure d'espace de Hilbert en étendant la hiérarchie algébrique de la bibliothèque Coquelicot, construite *via* le mécanisme des structures canoniques [143]. Nous formalisons un ensemble de concepts géométriques liés aux espaces de Hilbert, comme le projeté orthogonal ou le complémentaire orthogonal d'un sous-espace, et démontrons certaines propriétés fondamentales de ces objets géométriques. Les équations aux dérivées partielles sont des équations impliquant des fonctions. Nous formalisons des espaces de Hilbert fonctionnels, et plus précisément l'espace des formes linéaires continues d'un espace de Hilbert.

Preuve formelle du théorème de Lax–Milgram : Nous démontrons formellement le théorème de Lax–Milgram, un résultat-clef permettant d'établir la convergence de la méthode des éléments finis. La démonstration du théorème de Lax–Milgram repose sur un ensemble conséquent de résultats géométriques dans les espaces de Hilbert, ainsi

que sur d'autres résultats intermédiaires importants, *e.g.* un théorème de point fixe de Banach ou le théorème de Riesz–Fréchet, un résultat de représentation des formes linéaires continues d'un espace de Hilbert *via* le produit scalaire.

Sous-espaces de dimension finie et propriétés de complétude : L'idée générale de la méthode des éléments finis est d'approcher le volume irrégulier par un maillage d'éléments réguliers. Ce maillage peut être abstrait mathématiquement comme un sous-espace de dimension finie du volume irrégulier, *i.e.* comme un sous-espace de dimension finie d'un espace de Hilbert. La preuve de convergence de la méthode repose sur deux applications du théorème de Lax–Milgram, sur l'espace de Hilbert d'une part et sur le sous-espace de dimension finie d'autre part. Pour ne pas formaliser deux versions du théorème, nous le généralisons pour tout sous-module complet d'un espace de Hilbert. Il est trivial que l'espace de Hilbert complet est un sous-module complet de lui-même. En revanche, nous avons dû formaliser les sous-espaces de dimension finie et démontrer leur complétude, plus difficile à établir et reposant sur de nombreux résultats topologiques.

Publications. *La formalisation des espaces de Hilbert et du théorème de Lax–Milgram a fait l'objet d'un article avec S. Boldo, F. Clément, V. Martin et M. Mayero [26] ainsi qu'à un résumé court avec les mêmes auteurs [27]. La formalisation des sous-espaces de dimension finie a fait l'objet de deux articles [68, 69].*

1.2.5 Conclusion

Enfin, le chapitre 9 conclut ce document en résumant les contributions de la thèse puis présente des perspectives de travail.

1.3 Liens vers les formalisations Coq

Dans ce manuscrit, les résultats dont la démonstration a été formalisée en Coq sont marqués du logo  . Lorsque la démonstration repose sur la logique classique ou des hypothèses de « décidabilité » qui ne sont pas intuitionnistes, le logo  est utilisé.

D'un point de vue mathématique, il est dit qu'un problème est décidable s'il existe un algorithme qui permette de le décider, *i.e.* de répondre au problème par oui ou par non. En Coq (sans ajout d'axiome supplémentaire), une proposition P telle qu'il existe une fonction booléenne qui soit équivalente à P correspond à un problème décidable. En d'autres termes, les fonction booléennes définissables forment un sous-ensemble des problèmes décidables. En revanche, l'ajout de certains axiomes rend définissables de nouvelles fonctions booléennes qui ne sont pas nécessairement décidables.

Dans ce manuscrit, nous utiliserons l'abus de langage « décidable » pour faire référence à toute proposition pour laquelle il existe une fonction booléenne définissable équivalente, même si cela provient de l'ajout d'un axiome. Nous utiliserons le terme d'*hypothèse de décidabilité* pour les hypothèses de la forme $\{P\} + \{\neg P\}$, mais aussi pour les hypothèses de la forme $P \vee \neg P$.

L'ensemble du code Coq, ainsi que les instructions permettant de compiler les formalisations, sont disponibles à l'adresse :

`https://www.lri.fr/~faissole/these_coq.html`

L'annexe A propose une correspondance entre les principaux résultats présentés dans le manuscrit et les énoncés Coq correspondants.

Chapitre 2

Prérequis fondamentaux

Ce manuscrit traite d'un sujet à l'intersection de trois domaines de recherche : l'arithmétique à virgule flottante, la preuve formelle interactive et l'analyse numérique. Chacun de ces trois domaines est attaché à un vocabulaire et des notations spécifiques qu'il convient d'introduire le plus précisément possible. Ce chapitre décrit les définitions et résultats fondamentaux que nous utiliserons dans l'ensemble du document.

En section 2.1, nous présentons les définitions et propriétés de base de l'arithmétique à virgule flottante. Puis, en section 2.2, nous décrivons brièvement les bibliothèques Coq sur lesquelles nous nous basons. En section 2.3, nous présentons les méthodes de Runge-Kutta utilisées pour résoudre numériquement les équations différentielles ordinaires.

Remarque. *Dans ce chapitre, ainsi que dans la suite du manuscrit,* $\mathbb{R}^+$ *est l'ensemble des nombres réels positifs,* $\mathbb{R}^*$ *est égal à* $\mathbb{R} \setminus \{0\}$ *et* $\mathbb{R}_+^*$ *est l'ensemble des nombres réels strictement positifs. De même,* $\mathbb{N}^*$ *désigne les entiers naturels non nuls.*

2.1 Arithmétique à virgule flottante

Dans cette section, nous présentons quelques définitions élémentaires d'arithmétique à virgule flottante et d'analyse d'erreurs d'arrondi.

2.1.1 Définition des nombres à virgule flottante

Les formats de représentation des nombres à virgule flottante ainsi que les opérations sur ces nombres sont régis par la norme IEEE-754, dont la première version remonte à 1985 [112] et qui a fait l'objet d'une révision majeure en 2008 [113]. Suivant le niveau de détails requis pour énoncer une définition ou une propriété, la norme IEEE-754 définit quatre couches d'abstraction dans la représentation des nombres flottants [113, §3.2].

- La première couche définit les nombres flottants comme les éléments de $\overline{\mathbb{R}}$ (*i.e.* $\mathbb{R} \cup \{-\infty, +\infty\}$), le format flottant considéré n'a donc pas d'incidence à ce niveau.
- La seconde couche définit les nombres flottants comme un ensemble fini de nombres réels, plus les valeurs exceptionnelles $-\infty, +\infty, +0, -0$ et NaN (Not-a-

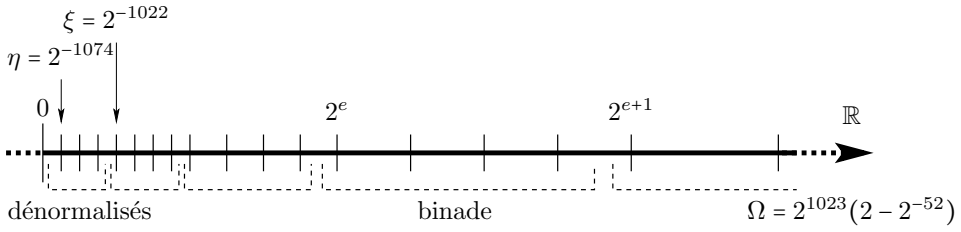


FIGURE 2.1 – Représentation de l'ensemble des flottants sur la droite des réels

Number). La valeur exceptionnelle NaN est utilisée pour encoder le résultat d'une opération invalide, *e.g.* $\frac{0}{0}$ ou $+\infty + (-\infty)$ et est absorbante. La norme IEEE-754 définit deux type de NaNs : les NaNs silencieux et les NaNs avertisseurs (voir [155, §3.1.6] pour plus de détails). Lorsqu'au moins l'un des opérandes d'un opérateur de comparaison est NaN, la valeur renvoyée est systématiquement `false`. En particulier, NaN = NaN renvoie `false` [1].

- Les troisièmes et quatrièmes couches d'abstraction décrivent respectivement la manière dont sont représentés les nombres flottants (triplets signe, mantisse, exposant) et quel est l'encodage de ces triplets comme suites de bits.

Dans le contexte de l'analyse d'erreurs, une description relativement haut niveau est suffisante. Dans cette section, nous donnons une définition des nombres flottants se rapprochant du second niveau d'abstraction de la norme IEEE-754. Nous ferons toutefois une confusion entre +0 et −0, le nombre flottant 0 étant considéré comme une valeur finie standard.

Un format flottant $\mathbb{F}$ est caractérisé par un tuple $(\beta, p, e_{\min}, e_{\max})$ où l'entier $\beta \geqslant 2$ est la base, $p \in \mathbb{N}^*$ est la précision, $e_{\min} \in \mathbb{Z}$ et $e_{\max} \in \mathbb{Z}$ sont les valeurs minimales et maximales de l'exposant. Un nombre flottant dans $\mathbb{F}$ est soit une valeur exceptionnelle parmi $+\infty, -\infty$ et NaN, soit une valeur finie égale à $\pm d_0.d_1 \cdots d_{p-1} \times \beta^e$ (avec d_i des chiffres dans la base β) et telle que $e_{\min} \leq e \leq e_{\max}$.

La norme IEEE-754 définit plusieurs formats d'arithmétique à virgule flottante [112, 113]. Parmi les formats flottants binaires (*i.e.* $\beta = 2$), les exemples les plus connus sont le format *binary32* (simple précision) qui est tel que $p = 24$ et le format *binary64* (double précision) qui est tel que $p = 53$. Depuis la révision majeure de la norme en 2008 [113], des formats flottants décimaux ($\beta = 10$) ont été spécifiés. Nous pouvons notamment citer les formats *decimal64* ($p = 16$) et *decimal128* ($p = 34$). La figure 2.1 présente la répartition des nombres flottants finis positifs sur l'axe réel dans le cas du format *binary64* ($\beta = 2$, $p = 53$). Les notations utilisées en figure 2.1 sont introduites ci-dessous (en section 2.1.2) dans le cas de formats flottants génériques.

1. Ainsi, pour tester si x est un NaN, le test qu'il est recommandé d'exécuter est $x = x$.

2.1.2 Nombres flottants normalisés et dénormalisés

Deux classes de nombres flottants finis

Parmi les nombres flottants finis, nous distinguons les nombres flottants dits normalisés et dénormalisés. Pour un format caractérisé par le tuple $(\beta, p, e_{\min}, e_{\max})$:

- les nombres flottants dits normalisés vérifient $d_0 \neq 0$ et $e_{\min} \leq e \leq e_{\max}$. Le plus petit nombre flottant normalisé positif est noté $\xi = \beta^{e_{\min}}$ et le plus grand nombre flottant normalisé est noté $\Omega = (\beta - \beta^{1-p})\beta^{e_{\max}}$. Un ensemble de nombres flottants binaires ayant le même exposant est appelé une binade. Pour les nombres flottants décimaux, nous parlerons de décade. En *binary64*, $\xi = 2^{-1022}$ et $\Omega \approx 2^{1024}$;
- les nombres flottants de valeurs absolues strictement inférieures à ξ sont dits dénormalisés et sont tels que $e = e_{\min}$ et $d_0 = 0$. Le plus petit nombre flottant dénormalisé positif non nul est noté $\eta = \beta^{e_{\min}-p+1}$. En *binary64*, $\eta = 2^{-1074}$.

Dépassements de capacité

Lorsque $e > e_{\max}$, il se produit ce que nous appelons un dépassement de capacité supérieur (*overflow* en anglais). Étant donnée l'absence de terme concis pour désigner les dépassements de capacité supérieurs en langue française, nous pourrons utiliser la terminologie de *dépassement supérieur*.

Lorsque nous considérons une valeur inférieure (en valeur absolue) au plus petit nombre flottant normalisé ξ, nous parlerons de dépassement de capacité inférieur (*underflow* en anglais). La terminologie concise *dépassement inférieur* pourra être utilisée dans ce document. Suivant le format flottant considéré, les nombres flottants dénormalisés peuvent être ou non ignorés. Lorsque les valeurs dénormalisées sont ignorées, nous parlons de dépassement inférieur abrupt. Lorsqu'au contraire, les nombres flottants dénormalisés sont pris en compte, comme l'impose la norme IEEE-754 [113], nous parlons de dépassement inférieur graduel [2].

Remarque. *Dans ce manuscrit, nous dirons parfois que les dépassements de capacité inférieurs ne sont pas* pris en compte*. Cela ne signifie pas que que les dépassements inférieurs sont considérés comme abrupts, mais que l'exposant n'est pas borné par l'exposant minimal* $e_{\min}$ *(alternativement, que* $e_{\min} = -\infty$*).*

Ulp d'un nombre

Une notion fondamentale en arithmétique à virgule flottante et jouant un rôle important en analyse d'erreurs est la notion d'*ulp* d'un nombre flottant. Ce terme provient

2. Notons, comme le font remarquer Muller *et al.* [155, §2.1], que les concepts de dépassements de capacité inférieurs et supérieurs sont ambigus (même dans la norme IEEE-754). Lorsqu'on dit qu'une fonction produit un *underflow*, cela peut signifier soit que le résultat exact est strictement plus petit que ξ en valeur absolue (*underflow* avant arrondi) soit que le résultat arrondi (sans prendre en considération la borne $e_{\min}$ sur l'exposant) est plus petit que ξ en valeur absolue (*underflow* après arrondi). La même remarque s'applique aux dépassements supérieurs, pour lesquels l'ambiguïté est moins subtile.

de l'abréviation du terme anglophone *unit in the last place*. L'*ulp* d'un nombre flottant positif correspond à l'espacement entre ce nombre et le nombre flottant consécutif immédiatement supérieur. La définition est symétrique pour les nombres flottants négatifs, *i.e.* l'*ulp* d'un nombre flottant négatif est l'écart entre ce nombre et le nombre flottant qui le précède. La communauté d'arithmétique des ordinateurs a successivement produit plusieurs définitions de la notion d'*ulp* [155], avec des variations pour les nombres flottants se trouvant à la frontière de deux binades (puissances de la base). La définition ici utilisée, qui étend la notion d'*ulp* à tout nombre réel, est généralement dénommée *ulp* de Goldberg [155, définition 2.6],[83]. Plus précisément, pour $|x| \in [\beta^e, \beta^{e+1}[$, si $x \neq 0$ alors $ulp(x) = \beta^{\max(e,e_{\min})-p+1}$ et $ulp(0) = 0$.

La notion d'*ulp* est corrélée à la notion de successeur d'un nombre flottant. Pour un nombre flottant $x \in \mathbb{F}$, le successeur de x est noté $\mathrm{succ}(x)$. Pour $x \in \mathbb{F}$ positif, la définition du successeur est simplement $\mathrm{succ}(x) = x + ulp(x)$. La définition du successeur d'un nombre flottant négatif est plus délicate, le cas où le nombre flottant en question est une puissance de la base devant être traité à part. Un moyen relativement simple d'étendre la notion de successeur aux nombres négatifs est de d'abord définir la notion de prédécesseur d'un nombre flottant positif t, noté $\mathrm{pred}(t)$ (qui dépend de la quantité $ulp(t)$, voir la définition dans l'ouvrage de Boldo et Melquiond [38, §1.1.2]). La notion de successeur est alors étendue aux nombres négatifs *via* la définition $\mathrm{succ}(x) = -\mathrm{pred}(-x)$.

2.1.3 Arrondis en arithmétique à virgule flottante

Le résultat exact d'une opération ou d'un algorithme n'est pas nécessairement représentable dans le format flottant $\mathbb{F}$ considéré. Il est alors possible d'utiliser une fonction $\circ : \mathbb{R} \to \mathbb{F}$ qui détermine vers quel nombre flottant arrondir le résultat obtenu.

Modes d'arrondi

La norme IEEE-754 spécifie plusieurs modes d'arrondi [113] :

- trois modes d'arrondi dirigés :
  - l'arrondi vers $-\infty$ noté $\circ_{\mathrm{RD}}$, qui arrondit un nombre réel x vers le plus grand nombre flottant inférieur ou égal à x,
  - l'arrondi vers $+\infty$ noté $\circ_{\mathrm{RU}}$, qui arrondit un nombre réel x vers le plus petit nombre flottant supérieur ou égal à x,
  - l'arrondi vers 0 noté $\circ_{\mathrm{RZ}}$, qui correspond à $\circ_{\mathrm{RD}}$ quand x est positif et à $\circ_{\mathrm{RU}}$ quand x est strictement négatif.
- l'arrondi au plus proche $\circ_{\mathrm{RN}}$, qui est le mode d'arrondi par défaut. Lorsque le nombre à arrondir est équidistant de deux nombres flottants (nous parlerons de *point-milieu*), il est nécessaire de choisir une règle pour déterminer vers lequel arrondir. Nous parlerons d'une *règle de bris d'égalité* (la terminologie anglophone *tie-breaking rule* étant plus connue). La norme définit deux règles pour $\circ_{\mathrm{RN}}$:
  - vers le nombre flottant le plus proche pair, l'arrondi est alors noté $\circ_{\mathrm{RN}}^{\mathrm{even}}$,
  - vers le nombre flottant de plus grande valeur absolue, l'arrondi est noté $\circ_{\mathrm{RN}}^{\mathrm{away}}$.

Nous ne donnons pas ici les détails du comportement des arrondis par rapport aux valeurs exceptionnelles et aux dépassements de capacité supérieurs (voir la norme IEEE-754 [113] pour plus de détails).

La majeure partie des résultats présentés dans ce manuscrit correspondent à un mode d'arrondi au plus proche. Par défaut, la notation $\circ$ fera référence à l'arrondi au plus proche, avec une règle de bris d'égalité quelconque. L'arrondi au plus proche d'un nombre réel x sera donc noté $\circ(x)$.

Arrondi correct

L'implémentation d'une fonction en arithmétique à virgule flottante satisfait la propriété d'arrondi correct si son résultat est égal à l'arrondi (suivant un mode d'arrondi donné) du résultat exact de la fonction. Plus formellement, l'arrondi correct est défini comme suit :

Définition 2.1 (Arrondi correct). *Soit $\mathbb{F}$ un format d'arithmétique à virgule flottante. Soit $n \in \mathbb{N}^*$, $f : \mathbb{R}^n \to \mathbb{R}$. Soit $\widetilde{f} : \mathbb{F}^n \to \mathbb{F}$ une implémentation de f dans le format $\mathbb{F}$. Soit $\circ : \mathbb{R} \to \mathbb{F}$ une fonction d'arrondi. On dit que $\widetilde{f}$ est correctement arrondie vis-à-vis de $\circ$ si :*

$$\forall x \in \mathbb{F}^n,\ \widetilde{f}(x) = \circ(f(x)).$$

La norme IEEE-754 impose que la propriété d'arrondi correct soit vérifiée pour six opérations élémentaires [113], à savoir l'addition, la soustraction, la multiplication, la division, la racine carrée et le `FMA` (Fused Multiply-Add). L'opération `FMA` permet de réaliser une multiplication suivie d'une sommation *via* un seul arrondi, *i.e.* :

$$\forall x, y, z \in \mathbb{F}, \texttt{FMA}(x, y, z) = \circ(x \times y + z).$$

Imposer la propriété d'arrondi correct permet de rendre déterministe le résultat des opérations élémentaires [130] et de faciliter les analyses d'erreurs d'arrondi. La norme IEEE-754 recommande l'implémentation correcte de certaines fonctions mathématiques, *e.g.* les fonctions trigonométriques [113, §9.2], plus difficile à obtenir. Le chapitre 4 de ce manuscrit est voué à l'exhibition d'un algorithme en arrondi correct pour le calcul de moyenne de deux nombres flottants décimaux.

Comme les opérations élémentaires sont en arrondi correct, nous pourrons utiliser les notations $\oplus$, $\ominus$, $\otimes$ et $\oslash$ pour désigner les opérations arrondies (avec un mode d'arrondi au plus proche avec règle quelconque de bris d'égalité) correspondant à $+, -, \times$ et $/$, de sorte que $x \oplus y = \circ(x + y), x \ominus y = \circ(x - y), x \otimes y = \circ(x \times y)$ et $x \oslash y = \circ(\frac{x}{y})$.

Nous utiliserons une notation compacte $\circ[\cdots]$ qui signifie que toutes les opérations à l'intérieur des crochets sont arrondies. En l'absence de parenthèses explicites, l'ordre d'évaluation des opérations à l'intérieur des crochets respecte les priorités usuelles des opérations, *e.g.* $\circ[ab + c] = (a \otimes b) \oplus c$. En revanche, les opérations flottantes n'étant pas nécessairement associatives, nous avons dû choisir une convention pour l'ordre d'évaluation d'opérations identiques : le parenthésage de la droite vers la gauche sera dans ce cas considéré comme implicite, *e.g.* $\circ[a + bc + d] = a \oplus ((b \otimes c) \oplus d)$.

2.1.4 Analyse des erreurs d'arrondi

Un premier concept clef en analyse d'erreurs est la distinction entre les erreurs dites absolues et relatives. L'erreur absolue commise lors de l'arrondi $\widetilde{x}$ d'un nombre réel x est égale à $\widetilde{x} - x$ tandis que l'erreur relative est égale à $\frac{\widetilde{x}-x}{x}$.

Donnons quelques résultats fondamentaux permettant de borner les erreurs absolues et relatives dans le cas de l'arrondi au plus proche. Dans l'ensemble de la section, nous ne tenons pas compte d'éventuels dépassements de capacité supérieurs. Un premier résultat, très général, permet de borner l'erreur d'arrondi *via* la notion d'*ulp* :

Lemme 2.1. *Soit $x \in \mathbb{R}$. Alors $|\circ(x) - x| \leqslant \frac{1}{2} ulp(x)$.*

Suivant que la valeur absolue de x soit supérieure à ξ ou non, le lemme 2.1 s'instancie comme résultat sur l'erreur relative ou absolue. Si $\xi \leqslant |x|$, le résultat suivant s'applique :

Lemme 2.2. *Soit $x \in \mathbb{R}$. Supposons que $\xi \leqslant |x|$. Alors $|\circ(x) - x| \leqslant u|x|$ avec $u = \frac{1}{2}\beta^{1-p}$.*

La quantité u, très fréquemment utilisée dans ce manuscrit, est appelée *unité d'arrondi*. Dans le cas du format *binary64*, $u = 2^{-53}$.

Lorsque $|x|$ est strictement inférieur au plus petit nombre flottant normalisé ξ, l'erreur relative commise en arrondissant x peut devenir très grande, mais nous pouvons prouver un résultat sur l'erreur absolue :

Lemme 2.3. *Soit $x \in \mathbb{R}$. Supposons que $|x| \leqslant \beta\xi + \frac{\eta}{2}$. Alors $|\circ(x) - x| \leqslant \frac{\eta}{2}$.*

La combinaison du lemme 2.2 et du lemme 2.3 permet de déduire un résultat unificateur, à savoir que pour tout $x \in \mathbb{R}$, $|\circ(x) - x| \leqslant \max\left(u|x|, \frac{\eta}{2}\right)$. Si cette borne s'avère fine, elle est difficile à manipuler et à propager. Nous utiliserons généralement un résultat plus faible, à savoir que pour tout $x \in \mathbb{R}$, $|\circ(x) - x| \leqslant u|x| + \frac{\eta}{2}$.

Jeannerod et Rump [119, théorème 2.1] ont exhibé une borne plus fine (et souvent optimale) se traduisant par pour tout $x \in \mathbb{R}$, en considérant la plage des exposants comme non bornée, $|\circ(x) - x| \leqslant \frac{u}{1+u}|x|$. Nous démontrons formellement une version de ce résultat qui tient compte d'éventuels dépassements de capacité inférieurs [3] :

Lemme 2.4.  *Soit $x \in \mathbb{R}$. Alors :*

$$|\circ(x) - x| \leqslant \frac{u}{1+u}|x| + \frac{\eta}{2}.$$

Dans la majorité des cas, le gain de précision obtenu en utilisant le lemme 2.4 peut paraître relativement faible. En fait, le principal intérêt de ce résultat est de simplifier certaines bornes d'erreurs (voir chapitres 5 et 6).

Nous démontrons une autre variante de ce résultat raffiné dans le cas où le nombre réel considéré est supérieur à ξ en valeur absolue. Dans ce cas, le terme en η est nul.

3. Le résultat reste valable en remplaçant $\frac{u}{1+u}|x| + \frac{\eta}{2}$ par $\max(\frac{u}{1+u}|x|, \frac{\eta}{2})$.

Lemme 2.5. *Soit $x \in \mathbb{R}$. Supposons que $\xi \leqslant |x|$. Alors :*

$$|\circ(x) - x| \leqslant \frac{u}{1+u}|x|.$$

Nous démontrons également que, dans le cas de l'addition ou de la soustraction de deux nombres flottants, le terme en η disparaît, *i.e.* :

Lemme 2.6. *Soit $x, y \in \mathbb{F}$. Soit $\diamond \in \{+, -\}$. Alors :*

$$|\circ(x \diamond y) - (x \diamond y)| \leqslant \frac{u}{1+u}|x \diamond y|.$$

Il s'agit de la conséquence d'une propriété de l'arithmétique à virgule flottante assurant que toute addition provoquant un dépassement de capacité inférieur est exacte [83].

Le modèle standard de l'arithmétique à virgule flottante est une abstraction définie par Higham [107] au regard des résultats d'analyse d'erreurs présentés ci-dessus. En tenant compte des dépassements graduels de capacité inférieurs, le modèle standard de l'arithmétique à virgule flottante prend la forme suivante [107, p. 56] [4] :

Abstraction 2.7. Modèle standard en tenant compte des dépassements graduels inférieurs
Soient $x, y \in \mathbb{F}$. Soit $\diamond \in \{+, -, \times, /\}$ une opération élémentaire. Alors :

- $\circ(x \diamond y) = (x \diamond y)(1+\varepsilon) + \delta$, *avec* $|\varepsilon| \leqslant u$, $|\delta| \leqslant \frac{\eta}{2}$ *et* $\varepsilon\delta = 0$;
- *Si* $\diamond \in \{+, -\}$, $\circ(x \diamond y) = (x \diamond y)(1+\varepsilon)$ *avec* $|\varepsilon| \leqslant u$.

Higham propose un ensemble de notations permettant d'établir des bornes lisibles et concises sur les erreurs d'arrondi d'algorithmes itérant des opérations élémentaires [107], tels que la sommation de nombres flottants ou le produit scalaire de deux vecteurs. Les notations θ et γ peuvent être caractérisées *via* le résultat suivant, dont la démonstration peut être consultée dans l'ouvrage de Higham [107, lemme 3.1, p. 63] :

Lemme 2.8. Accumulation d'erreurs
Soit $n \in \mathbb{N}^*$ tel que $nu < 1$. Soient $\varepsilon_1, \ldots, \varepsilon_n \in \mathbb{R}$ tels que pour tout i, $|\varepsilon_i| \leqslant u$. Alors :

$$\exists \theta_n \in \mathbb{R}, \prod_{i=1}^{n}(1+\varepsilon_i) \leqslant (1+\theta_n) \text{ avec } |\theta_n| \leqslant \gamma_n := \frac{nu}{1-nu}.$$

À partir du modèle standard de l'arithmétique à virgule flottante (abstraction 2.7) et du lemme 2.8, il est possible de déduire une borne sur l'erreur d'arrondi commise lors d'une sommation de nombre flottants, qui reste valable en présence de dépassements graduels de capacité inférieurs [5] (démontré en Coq par Roux [177, théorème 1.8]) :

4. Ce résultat reste valable en remplaçant la quantité u par $\frac{u}{1+u}$.
5. L'absence de terme en η est une conséquence du caractère exact des additions provoquant des dépassements de capacité inférieurs [83] (voir lemme 2.6).

Théorème 2.9. Borne sur l'erreur d'arrondi d'une sommation
Soit $n \in \mathbb{N}^*$ tel que $nu < 1$. Soient $x_1, \ldots, x_n \in \mathbb{F}$. Soit $s = \sum_{i=1}^{n} x_i$ et soit $\widetilde{s}$ une implémentation de s dans le format $\mathbb{F}$ avec un ordre quelconque d'évaluation des sommations. Alors :

$$|\widetilde{s} - s| \leqslant \gamma_{n-1} \sum_{i=1}^{n} |x_i|.$$

Plus récemment, Jeannerod et Rump ont proposé une amélioration de ce résultat [119, théorème 4.1] en exhibant une borne plus fine, plus concise et ne nécessitant pas d'hypothèse sur la taille de la sommation :

Théorème 2.10. Borne raffinée sur l'erreur d'arrondi d'une sommation
Soit $n \in \mathbb{N}^*$. Soient $x_1, \ldots, x_n \in \mathbb{F}$. Soit $s = \sum_{i=1}^{n} x_i$ et soit $\widetilde{s}$ une implémentation de s dans le format $\mathbb{F}$ avec un ordre quelconque d'évaluation des sommations. Alors, en tenant compte des dépassements graduels inférieurs :

$$|\widetilde{s} - s| \leqslant \frac{(n-1)u}{1+u} \sum_{i=1}^{n} |x_i| \leqslant (n-1)u \sum_{i=1}^{n} |x_i|.$$

Ce résultat s'inscrit dans la lignée de travaux débutés depuis 2008 par Rump *et al.* [179, 180, 118, 119] pour raffiner les bornes d'erreurs des ouvrages de référence (*e.g.* [107]). L'idée générale est de se passer des γ_n et de se ramener à un terme d'erreur linéaire en u, plus lisible et permettant de réduire les hypothèses sur la dimension du système considéré, *e.g.* la condition $nu < 1$ permettant de définir $\gamma_n = \frac{nu}{1-nu}$.

Nous avons présenté un ensemble de notions et résultats utiles pour mener l'analyse des erreurs d'arrondi. Nous allons maintenant présenter les bibliothèques Coq sur lesquelles reposent nos formalisations.

2.2 Preuves formelles avec Coq

Dans cette section, nous présentons brièvement les bibliothèques Coq que nous utilisons pour formaliser les résultats de ce manuscrit.

2.2.1 Bibliothèque standard des réels

La majorité des bibliothèques Coq que nous utilisons sont basées sur la bibliothèque standard des nombres réels de Coq, principalement développée par Mayero en 2001 [148] et étendue par Desmettre l'année suivante [60]. Contrairement à d'autres bibliothèques d'analyse basées sur une construction des nombres réels, *e.g.* la bibliothèque MathClasses basée sur des suites de Cauchy [125] ou la bibliothèque standard de Mizar utilisant des coupures de Dedekind [200], le type des nombres réels de la bibliothèque standard de Coq, noté `R`, repose sur une axiomatisation des opérations et propriétés élémentaires de $\mathbb{R}$. Les réels de la bibliothèque standard comportent deux éléments `R0` et `R1` ainsi que les opérations d'addition, d'opposé, de soustraction, de multiplication

et de division. À cela s'ajoute une relation d'ordre stricte `Rlt` dont dérive la relation d'ordre large `Rle`. Les notations usuelles comme 0, 1, +, $*$, $\leqslant$ ou $<$ sont fournies par la bibliothèque standard de Coq et seront utilisées dans ce manuscrit. Certains des axiomes permettent de donner aux opérations élémentaires leur sens usuel. Nous dénombrons ainsi 14 axiomes élémentaires, dont voici quelques exemples :

```
Axiom Rplus_assoc :
        forall r1 r2 r3:R, r1 + r2 + r3 = r1 + (r2 + r3).
Axiom Rplus_0_l : forall r:R, 0 + r = r.
Axiom Rmult_comm : forall r1 r2:R, r1 * r2 = r2 * r1.
Axiom R1_neq_R0 : 1 <> 0.
Axiom Rlt_trans : forall r1 r2 r3:R, r1 < r2 → r2 < r3 → r1 < r3.
(* ... *)
```

La bibliothèque standard comporte trois autres axiomes, l'un des plus utilisés dans ce document étant `total_order_T`, qui munit les réels d'une relation d'ordre total, assurant ainsi la décidabilité de l'égalité entre deux nombres réels :

```
Axiom total_order_T : forall r1 r2:R, {r1 < r2} + {r1 = r2} + {r1 > r2}.
```

Enfin, les deux axiomes restants assurent respectivement que les réels sont archimédiens et que tout sous-espace non vide (de type `R -> Prop`) admet une borne supérieure.

La bibliothèque standard des réels comporte également une formalisation de l'ensemble `posreal` des réels strictement positifs, qui apparaît régulièrement dans ce document. Le type `posreal` est construit comme un type enregistrement (`Record`) à deux champs, le premier étant un nombre réel et le second une preuve de positivité de ce nombre :

```
Record posreal : Type := mkposreal {pos :> R; cond_pos : 0 < pos}.
```

Il est par ailleurs possible de raisonner sur des résultats d'analyse réelle, *e.g.* liés à des questions de différentiabilité ou d'intégrabilité (au sens de Riemann), en utilisant les réels de la bibliothèque standard. Cependant, comme remarqué par Boldo, Lelay et Melquiond [35, 134], il est aujourd'hui généralement admis que la bibliothèque standard des réels n'est pas adaptée à la formalisation et la preuve de résultats pointus d'analyse mathématique. Elle repose sur des fonctions partielles et sur l'usage intensif de types dépendants, qui rendent parfois les démonstrations fastidieuses. Prenons l'exemple classique de la dérivabilité de fonctions. Supposons que f et g sont deux fonctions dérivables. Dans la bibliothèque standard, pour calculer le point dérivé d'une fonction en un point x, il faut nécessairement fournir une preuve de dérivabilité de la fonction en x. Ainsi, pour obtenir la dérivée de $f + g$ en x, nous ne pouvons pas simplement utiliser la dérivabilité de f et g puis réécrire $(f + g)'(x) = f'(x) + g'(x)$. En effet, il faut d'abord démontrer la dérivabilité de $f + g$ en x pour pouvoir construire $(f + g)'(x)$. Cela rend les calculs de dérivées particulièrement fastidieux.

2.2.2 Bibliothèque Coquelicot d'analyse réelle

Pour pallier les défauts de la bibliothèque standard des réels, Boldo, Lelay et Melquiond ont développé la bibliothèque Coquelicot [35, 134]. Coquelicot est une extension conservative de la bibliothèque standard des réels qui permet d'une part de faciliter le développement de théories pointues d'analyse réelle et d'autre part de raisonner sur des espaces plus généraux. Coquelicot, et plus précisément l'ensemble des notions topologiques qui y sont développées, sert de base aux formalisations présentées dans les chapitres 7 et 8 de ce manuscrit. La présentation de la bibliothèque Coquelicot est par conséquent détaillée au début du chapitre 7.

2.2.3 Bibliothèque Mathematical Components

La bibliothèque Mathematical Components (ou MathComp) [144] regroupe un ensemble varié de formalisations Coq de théories mathématiques, comme l'analyse matricielle ou la théorie des groupes finis. Le projet trouve ses origines dans la preuve formelle du théorème des quatre couleurs [85] et sert de base à la démonstration Coq du théorème de Feit-Thompson [86]. MathComp utilise extensivement Ssreflect [87], un langage de tactiques Coq conçu comme une amélioration du langage standard de tactiques de Coq. Ssreflect permet le développement de preuves très concises et s'avère particulièrement adapté à la démonstration formelle de théorèmes reposant sur un socle mathématique conséquent. La communauté d'utilisateurs de Ssreflect est incitée à utiliser des lemmes dits de *réflection* (ou *vues*) qui permettent de transformer une proposition (de type `Prop`) en un prédicat booléen (de type `bool`) afin de faciliter les raisonnements.

Les développements Coq présentés dans les chapitres 5 à 6 de ce document n'utilisent à ce jour qu'un ensemble limité de fonctionnalités de la bibliothèque Mathematical Components, à savoir la formalisation des vecteurs et matrices et le mécanisme des grands opérateurs (*big operators*) [18]. Nous présentons ici une description « haut niveau » de ces notions. Pour une présentation plus détaillée de Ssreflect (respectivement de MathComp), voir [87] (respectivement [144]).

Vecteurs et matrices dans Mathematical Components

Mathematical Components comporte une bibliothèque pour la manipulation de vecteurs et matrices. Très complète, cette bibliothèque définit notamment les opérations entre matrices ainsi que les notions de déterminant, de trace et de matrice transposée. Elle permet de manipuler les lignes ou les colonnes d'une matrice avec par exemple la possibilité d'opérer des permutations. Il est également possible d'extraire une sous-matrice et de faire des décompositions de matrices par blocs.

Le type des matrices réelles de taille $m \times n$ est noté `'M[R]_(m, n)` et défini comme :

```
Inductive matrix := Matrix of {ffun 'I_m * 'I_n → R}.
```

Le mot-clef `ffun` permet de définir une fonction finie [144, §6.5], c'est-à-dire une fonction dont le domaine est fini. Ici, l'ensemble de départ de la fonction est `'I_m * 'I_n`

où `'I_m` et `'I_n` sont respectivement les ensembles d'entiers naturels strictement inférieurs à `m` et à `n`. Ainsi, si nous définissons une matrice `A : 'M[R]_(n, n)` et que nous considérons `i, j : 'I_n`, alors il suffira d'écrire `A i j` pour accéder à l'élément de `A` d'indices `i` et `j`.

Dans le cas des matrices carrées de taille $n \times n$, il est possible d'utiliser une notation condensée pour réduire la notation du type `'M[R]_(n, n)` en `'M[R]_n`. De même, pour le type `'M[R]_(n, 1)` des vecteurs colonnes de taille n, il est possible d'utiliser la notation `'cV[R]_n` et pour le type `'M[R]_(1, n)` des vecteurs lignes de taille n, il est possible d'utiliser la notation `'rV[R]_n`.

Il est possible de définir une matrice de type `'M[R]_(m, n)` à partir d'une fonction `F : 'I_m -> 'I_n -> R` en utilisant un opérateur `\matrix_(i,j) (F i j)`. Par exemple, le produit de deux matrices de tailles $m \times n$ et $n \times p$ est défini comme [6] :

```
Definition mulmx {m n p} (A : 'M_(m, n)) (B : 'M_(n, p)) : 'M[R]_(m, p)
                         := \matrix_(i, j) \sum_(k < n) (A i k * B k j).
```

Le produit de deux matrices `A : 'M_(m, n)` et `B : 'M_(n, p)` est noté `A *m B`. Un point intéressant justifiant l'utilisation des vecteurs comme instances de matrices est le fait que Coq soit capable d'inférer que le produit d'une matrice carrée de taille $n \times n$ et d'un vecteur colonne de taille n soit un vecteur colonne de taille n. L'opérateur `\sum` ci-dessus est une sommation au sens des grands opérateurs (voir paragraphe suivant).

Grands opérateurs

Les grands opérateurs [144, §5.7], [18] permettent d'itérer une opération élémentaire sur un ensemble de valeurs. D'un point de vue purement mathématique, il pourra par exemple s'agir de la sommation $\sum$ vis-à-vis de l'addition, de $\bigcup$ vis-à-vis de l'union ou de $\prod$ vis-à-vis de la multiplication.

Le mécanisme des grands opérateurs permet d'itérer une opération binaire sur différentes structures de données, *e.g.* une liste de valeurs ou l'ensemble `'I_n` pour $n \in \mathbb{N}$. Dans ce document, nous utiliserons les grands opérateurs pour décrire des opérations vectorielles et matricielles. Notre cas d'utilisation des grands opérateurs se limite donc à l'itération sur `'I_n`. Nous ne présenterons donc que cet aspect.

Soit $\diamond : \mathbb{R} \to \mathbb{R} \to \mathbb{R}$ une opération binaire dont l'élément neutre est noté $0_\diamond$. Le grand opérateur associé à $\diamond$ est noté $\bigdiamond$, $\bigdiamond_{i<n} v_i$ étant égal à $(v_0 \diamond (v_1 \diamond \cdots \diamond (v_{n-2} \diamond v_{n-1}) \dots))$ (pour $v \in \mathbb{R}^n$) et $\bigdiamond_{i<0} v_i$ étant égal à $0_\diamond$. Les opérations $\diamond$ sont donc évaluées de la droite vers la gauche. Le mécanisme Coq associé est utilisé *via* la notation compacte suivante :

```
\big[diamond / 0_diamond]_(i < n) (v i).
```

Étant donné l'ordre des évaluations, lorsque $\diamond$ est une opération quelconque (en particulier non-associative), le grand opérateur ne peut être déroulé que par la gauche, *i.e.* $\bigdiamond_{i<n+1} v_i = v_0 + \bigdiamond_{i<n} v_{i+1}$. En revanche, si $\diamond$ est une opération associative, il devient possible de réécrire $\bigdiamond_{i<n+1} v_i = (\bigdiamond_{i<n} v_i) + v_n$.

6. Comme le type de retour de la fonction est `'M[R]_(m, p)`, Coq est en mesure d'inférer le type des éléments de A et B ; il n'est donc pas nécessaire de préciser que ce sont des matrices sur $\mathbb{R}$.

2.2.4 Arithmétique des ordinateurs en Coq

Bibliothèque Flocq

Les chapitres 4 à 6 du manuscrit décrivent la formalisation en Coq de résultats d'arithmétique à virgule flottante. Ces formalisations reposent sur Flocq, une bibliothèque Coq d'arithmétique des ordinateurs développée par Boldo et Melquiond [37, 38]. Flocq permet de manipuler les nombres flottants suivant différents niveaux d'abstraction [38, §3], ces niveaux étant inspirés des couches d'abstraction de la norme IEEE-754 [113] (voir section 2.1.1).

Nous nous intéressons plus particulièrement à la représentation abstraite des nombres à virgule flottante. Les nombres dans un format d'arithmétique donné (*e.g. binary32* ou *decimal64*) sont définis comme un sous-ensemble des nombres réels `R` de la bibliothèque standard (en tant que prédicats de type `R -> Prop`). Plusieurs formats sont disponibles, incluant des formats d'arithmétique à virgule fixe ou flottante.

Les formats flottants permettent de caractériser des nombres de la forme $m\beta^e$ (avec m et e des entiers, la base β étant un entier strictement supérieur à 1).

La bibliothèque Flocq inclut un prédicat `generic_format : R -> Prop` dont les formats flottants usuels constituent des instances. Ce prédicat dépend de la base β et d'une fonction $\varphi : \mathbb{Z} \to \mathbb{Z}$ que nous appellerons *fonction d'exposant* (c'est cette fonction qui permettra d'instancier le prédicat avec le format souhaité). Nous ne rentrons pas dans les détails de la définition de ce prédicat mais en présentons l'idée générale. Un nombre $x \in \mathbb{R}$ satisfait le prédicat `generic_format` s'il vérifie :

$$x = \left\lfloor x\beta^{-\varphi(\mathrm{mag}_\beta(x))} \right\rfloor \times \beta^{\varphi(\mathrm{mag}_\beta(x))}$$

Dans la formule ci-dessus, $\mathrm{mag}_\beta(x) = \left\lfloor \log_\beta(x) \right\rfloor + 1$ permet d'obtenir l'entier e tel que $x \in [\beta^{e-1}, \beta^e[$, que Boldo et Melquiond [38] désignent comme la *magnitude* de x.

L'instance la plus simple de `generic_format` est `FLX`, qui correspond à des nombres flottants sans bornes sur les exposants, *i.e.* les dépassements de capacité ne sont pas pris en compte. En plus de β, le format `FLX` dépend de la précision p, les nombres flottants du format `FLX` étant de la forme $m\beta^e$ avec $|m| < \beta^p$. Dans Flocq, `FLX` peut être défini comme une instance de `generic_format` avec $\varphi = e \mapsto e - p$ (noté `FLX_exp` en Coq).

Le format `FLT` correspond pour sa part à des nombres flottants tenant compte d'éventuels dépassements graduels inférieurs mais ne tenant pas compte des dépassements supérieurs. Le format `FLT` dépend de la précision p et d'un exposant minimal $e_{\min}$ avec les contraintes $|m| < \beta^p$ et $e_{\min} - p + 1 \leqslant e$. Il s'agit d'une instance du prédicat `generic_format` avec $\varphi = e \mapsto \max(e - p, e_{\min} - p + 1)$ (noté `FLT_exp` en Coq).

Pour représenter les nombres de la forme $m\beta^e$, il est également possible d'utiliser un type enregistrement (type `Record`), nommé `float` et défini comme :

```
Record float (beta : radix) := Float { Fnum : Z ;
                                       Fexp : Z }.
```

Dans le code ci-dessus, `Float` est le nom de constructeur associé au type enregistrement `float`. `Fnum` correspond à m et `Fexp` correspond à e. Pour plonger un élément de type `float` dans les nombres réels, il faut utiliser l'opérateur `F2R` suivant :

```
Definition F2R {beta : radix} (f : float beta) :=
        IZR (Fnum f) * bpow beta (Fexp f).
```

Dans le code Coq ci-dessus, `IZR` permet de plonger les entiers relatifs dans les réels et `bpow beta (Fexp f)` est l'élévation de `beta` à la puissance `(Fexp f)`, *i.e.* β^e.

Flocq inclut des définitions alternatives des formats `FLX` et `FLT` qui sont basées sur le type `float` et définies *via* un type inductif à un seul constructeur [7]. Elles sont prouvées équivalentes aux définitions précédentes :

```
Inductive FLX_format (x : R) : Prop := FLX_spec (f:float spec) :
    x = F2R f → Zabs (Fnum f) < Zpower beta prec → FLX_format x.

Inductive FLT_format (x : R) : Prop := FLT_spec (f:float spec) :
    x = F2R f → Zabs (Fnum f) < Zpower beta prec
              → emin <= Fexp f → FLT_format x.
```

Pour un format flottant donné, différents modes d'arrondi sont disponibles (*e.g.* l'arrondi vers $+\infty$ ou vers 0). Le mode d'arrondi est déterminé par une fonction `rnd : R -> Z` permettant d'obtenir la mantisse du nombre flottant obtenu par arrondi, l'exposant étant quant à lui égal à $\varphi(\mathrm{mag}_\beta(x))$ lorsque x est un nombre flottant. Dans la suite de ce document, nous utiliserons principalement l'arrondi au plus proche, dont la fonction `rnd` correspondante est appelée `Znearest` et dépend de la règle de bris d'égalité `choice : Z -> bool`. Il est possible de travailler avec un arrondi au plus proche générique, `choice` étant laissé en paramètre. Cela nous permet de ne faire les preuves formelles qu'à une seule reprise, quelle que soit la règle de bris d'égalité.

La bibliothèque Flocq contient un nombre important de résultats, *e.g.* sur l'appartenance de nombres réels à un format donné, sur l'*ulp* ou sur les erreurs commises lorsque des nombres réels sont arrondis.

Outil Gappa

Gappa est un outil permettant de vérifier des propriétés arithmétiques de programmes numériques simples [57, 61]. Il est particulièrement adapté pour prouver automatiquement, sous certaines hypothèses, des bornes sur les erreurs d'arrondi.

Prenons l'exemple du calcul d'un coefficient dans un pas d'itération de la méthode d'Euler (voir sections 2.3.3 et 5.4.2). Le programme prend en argument un nombre $h \in \mathbb{F}$ tel que $2^{-60} \leqslant h \leqslant 1$ et un nombre $\ell \in \mathbb{R}$ tel que $-2 \leqslant h\ell \leqslant -2^{-100}$. Le programme calcule le produit de h et de ℓ. L'objectif est de borner l'erreur d'arrondi commise lors de ce calcul dans le format *binary64* avec mode d'arrondi au plus proche.

7. Entre Flocq et la norme IEEE-754, il y a un décalage de $p-1$ pour l'exposant `emin`, d'où la condition `emin <= Fexp f` dans le code Coq.

Le script Gappa correspondant est le suivant :

```
@rnd = float<ieee_64,ne>;
lt = rnd(l);
rf rnd= h*lt;
r = h*l;

{ h*l in [-2,-1b-100] /\ h in [1b-60,1]
   -> rf -/ r in [-2049b-63,2049b-63] /\ rf - r in [-4b-53, 4b-53]
}
```

Dans le code ci-dessus, `float<ieee_64,ne>` signifie que le format choisi est le format *binary64* avec mode d'arrondi au plus proche et prise en compte de potentiels dépassements ce capacité inférieurs. La variable `lt` est l'arrondi au plus proche de ℓ, *i.e.* $\circ(l)$. La variable `r` est le résultat du calcul exact de $h\ell$ et `rf` est le résultat arrondi, *i.e.* $\circ(h \times \circ(l))$.

Les hypothèses données à Gappa sont les intervalles de valeurs possibles pour h et $h\ell$, *i.e.* `h*l in [-2,-1b-100]` et `h in [1b-60,1]`.

Nous souhaitons prouver deux propriétés. La première est une borne sur l'erreur relative commise lors du calcul de `r`, *i.e.* `rf -/ r`. Nous souhaitons prouver que l'erreur est bornée par $2049 \times 2^{-63} \leqslant 0{,}01u$. La seconde propriété est une borne sur l'erreur absolue `rf - r`, de l'ordre de $4u$. À partir des hypothèses, Gappa valide (ou infère automatiquement) ces bornes.

Du point de vue de la preuve interactive, un intérêt de Gappa est la possibilité de générer un terme de preuve à partir d'un script Gappa, terme qui peut être vérifié par Coq. Il existe une tactique automatique `gappa` permettant de décharger à Gappa certaines étapes des preuves formelles que nous menons dans Coq.

Un outil similaire a été proposé par Magron, Constantinides et Donaldson [139] pour borner les erreurs d'arrondi absolues de programmes non-linéaires. L'outil est basé sur des techniques d'optimisation et de programmation semi-définie. Nous n'avons à ce jour pas essayé d'utiliser cet outil dans le cadre de nos problématiques.

Nous souhaitons plus particulièrement formaliser les erreurs d'arrondi de méthodes de résolution d'équations différentielles, qui sont présentées dans la section suivante.

2.3 Analyse numérique des équations différentielles

Dans cette section, nous présentons brièvement quelques méthodes numériques de résolution approchée d'équations différentielles ordinaires. Nous nous attachons plus particulièrement à présenter la classe des méthodes de Runge-Kutta, méthodes sur lesquelles portent les chapitres 5 à 6 de ce manuscrit.

2.3.1 Équations différentielles et problèmes de Cauchy

Nous nous intéressons aux méthodes de résolution numériques d'équations différentielles ordinaires du premier ordre. Ces équations ont pour inconnue une fonction $y : \mathbb{R} \to \mathbb{R}^d$ ($d \in \mathbb{N}^*$ étant la dimension du système) et sont caractérisées par une relation entre y et sa dérivée première. Nous ne donnons ici qu'une intuition sur les équations différentielles ordinaires. L'ouvrage de Demailly [59] contient une description détaillée de ces équations et de leur résolution numérique.

Dans le contexte de ce manuscrit, les équations sont de la forme :

$$y' = f(t, y), \qquad t \in \mathbb{R}, \quad y, y' \in \mathbb{R} \to \mathbb{R}^d, \quad f \in \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d.$$

Nous appelons *problème de Cauchy* (ou problème avec condition initiale) une équation différentielle ordinaire du premier ordre enrichie d'une *condition initiale*, *i.e.* une condition de la forme $y(t_0) = y_0 \in \mathbb{R}^d$ pour un certain $t_0 \in \mathbb{R}$. Pour peu que f réunisse certaines hypothèses, le théorème de Cauchy-Lipschitz assure l'existence et l'unicité de la solution d'un problème de Cauchy [59].

Le cas particulier le plus simple d'équations différentielles est celui des systèmes linéaires matriciels ($\mathbb{R}^{d\times d}$ désignant les matrices carrées de taille $d \in \mathbb{N}^*$) :

$$y' = Ay, \qquad A \in \mathbb{R}^{d\times d}.$$

La résolution directe des équations différentielles linéaires nécessite la construction d'une exponentielle de matrices, dont l'implémentation est connue comme difficile [84, 151]. Pire encore, dans le cas des systèmes non-linéaires, la résolution directe s'avère impossible. Ces difficultés sont la raison d'être de méthodes itératives permettant de résoudre numériquement certaines classes de problèmes de Cauchy. Les sections 2.3.2 et 2.3.3 sont vouées à la description de certaines de ces méthodes.

2.3.2 Méthodes numériques à un pas et multi-pas

Il existe une kyrielle de méthodes permettant d'approcher la solution d'équations différentielles ordinaires [98, 203, 59]. Le point commun de la majorité de ces méthodes est de permettre l'obtention d'une approximation de l'évaluation de la fonction inconnue sur un ensemble fini de valeurs réparties sur un intervalle d'intégration prédéfini.

Concrètement, cela suppose le choix d'un intervalle d'intégration $[t_0\ ;\ t_n]$ avec $t_0, t_n \in \mathbb{R}^+$ et $t_0 < t_n$. Cet intervalle est ensuite subdivisé en un ensemble de valeurs $t_0, t_1, \ldots, t_n$, la distance entre ces points pouvant être choisie constante ou non. Le principe des méthodes numériques itératives est de fournir, pour chaque point t_i ($0 \leqslant i \leqslant n$), une approximation $y_i \simeq y(t_i)$ de l'image de la fonction inconnue y en t_i. Ces approximations sont généralement obtenues à partir de la condition initiale $y_0 = y(t_0)$.

Nous distinguons les méthodes à un pas et les méthodes multi-pas :

- Les méthodes à un pas sont telles que pour tout $0 < i \leqslant n$, y_i est calculée à partir de la seule donnée de l'itération précédente y_{i-1}. Les méthodes de Runge-Kutta sont des méthodes numériques à un pas [45, 59].

- Dans le cas des méthodes multi-pas, y_i est calculée à partir de plusieurs itérations précédentes, *e.g.* y_{i-1}, y_{i-2} et y_{i-3} pour une méthode à 3 pas. Les méthodes d'Adam-Bashforth [59] constituent des exemples de méthodes multi-pas.

L'écart entre deux points d'abscisse t_{i-1} et t_i de l'intervalle d'intégration est appelé pas d'intégration à l'itération i. Le pas d'intégration peut être choisi constant, *i.e.* il existe $h > 0$ tel que pour tout i, $h = t_i - t_{i-1}$, ou bien variable, *i.e.* à chaque itération de la méthode correspond un pas d'intégration h_i.

Dans la suite de ce document, nous nous intéresserons plus particulièrement à des méthodes à un pas dont le pas d'intégration est constant.

2.3.3 Méthodes de Runge-Kutta

Parmi les méthodes numériques à un pas, la plupart appartiennent à la classe des méthodes de Runge-Kutta, nommées ainsi à la suite de deux articles fondateurs, l'un rédigé par Runge en 1895 [181], le second par Kutta en 1901 [128].

Tableau de Butcher

Sur la donnée du problème de Cauchy défini par $y' = f(t, y)$ et $y(t_0) = y_0$, les méthodes de Runge-Kutta sont caractérisées par la relation de récurrence suivante :

$$\forall n, y_{n+1} = y_n + h\Phi(t_n, y_n, h) = y_n + h\sum_{i=1}^{s} b_i k_i, \tag{2.1}$$

avec k_i défini comme :

$$k_i = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij} k_j\right). \tag{2.2}$$

Les coefficients c_i, a_{ij} et b_i, avec $i, j = 1, 2, \cdots, s$, ainsi que le *nombre d'étages* s, caractérisent donc à eux seuls une méthode de Runge-Kutta. Ainsi, une méthode de Runge-Kutta est généralement définie *via* son *tableau de Butcher* [45, 98], qui contient la valeur des coefficients :

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} \quad \equiv \quad \begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b} \end{array} .$$

La figure 2.2 présente quelques exemples de tableaux de Butcher associés à des méthodes de Runge-Kutta variées.

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

(a) RK4

$$\begin{array}{c|cc} \frac{1}{2}-\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\ \frac{1}{2}+\frac{\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

(b) Gauss-Legendre

FIGURE 2.2 – Tableaux de Butcher de méthodes de Runge-Kutta explicites et implicites

Méthodes explicites et implicites

Selon la forme de la matrice $\mathbf{A}$ contenant les coefficients a_{ij}, nous distinguons plusieurs sous-classes de méthodes de Runge-Kutta :

- Les méthodes *explicites*, comme la méthode d'Euler ou la méthode de Runge-Kutta d'ordre 4 définie par la figure 2.2(a), qui sont telles que le calcul des coefficients k_i dépend uniquement des k_j précédents, *i.e.* tels que $j < i$;
- Les méthodes *implicites*, comme la méthode de Runge-Kutta implicite d'ordre 4 par formule de Gauss-Legendre définie par la figure 2.2(b), sont telles que le calcul d'un coefficient k_i dépend de la résolution d'un système d'équations non-linéaires impliquant l'ensemble des coefficients k_j pour $j = 1, 2, \cdots, s$.

Dans la partie II de ce manuscrit, nous allons plus particulièrement borner les erreurs d'arrondi de méthodes de Runge-Kutta explicites.

Ordre d'une méthode de Runge-Kutta

L'*ordre* d'une méthode de Runge-Kutta est l'entier p tel que l'*erreur locale de méthode*, *i.e.* la distance entre la solution exacte $y(t_n)$ et la solution obtenue par la méthode de Runge-Kutta y_n est telle que

$$y(t_n) - y_n = O\left(h^{p+1}\right).$$

L'ordre d'une méthode de Runge-Kutta donne donc une indication de l'ordre de magnitude de l'*erreur de méthode* associée à un pas de l'intégration numérique. Si une méthode d'ordre élevé assure l'obtention d'un résultat théoriquement plus précis, cela va généralement de pair avec un nombre plus important de calculs flottants pouvant conduire à des erreurs d'arrondi plus importantes.

Pour les méthodes explicites d'ordre inférieur ou égal à 4, l'ordre p est égal au nombre d'étages s. En revanche, au-delà de l'ordre 4, l'ordre d'une méthode explicite peut être inférieur au nombre d'étages.

Méthodes de Runge-Kutta explicites classiques

Dans cette section, nous présentons trois méthodes de Runge-Kutta explicites classiques, qui serviront d'exemples dans dans les chapitres 5 et 6.

- **Méthode d'Euler explicite (RK1)**

La méthode d'Euler explicite est l'exemple le plus ancien de méthode numérique de résolution d'équations différentielles ordinaires. L'approche remonte en effet à des travaux d'Euler datant de 1768 [66], bien avant l'élaboration des méthodes de Runge-Kutta [181, 128] dont la méthode d'Euler est une instance. La méthode d'Euler utilise l'équation de la tangente au début du pas d'intégration, qui lie la fonction et sa dérivée.

Le tableau de Butcher associé à la méthode d'Euler est relativement simple (voir figure 2.3) et permet d'extraire la caractérisation suivante :

$$k_1 = f(t_n, y_n) \qquad y_{n+1} = y_n + hk_1 = y_n + hf(t_n, y_n).$$

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

FIGURE 2.3 – Tableau de Butcher de la méthode d'Euler

La méthode d'Euler s'avère assez peu précise, notamment lorsque le pas d'intégration choisi est relativement grand. Elle est particulièrement inadaptée lorsque la fonction inconnue croît/décroît fortement au cours d'un pas d'intégration, puisque la dérivée n'est évaluée qu'au début du pas d'intégration.

- **Méthode du point-milieu (RK2)**

Des méthodes plus élaborées ont été conçues pour pallier ce problème. Un premier exemple est la méthode RK2, ou méthode du « point-milieu » car la dérivée est également évaluée au milieu de l'intervalle d'intégration. Le tableau de Butcher associé à la méthode RK2 est décrit par la figure 2.4 et la relation de récurrence obtenue est :

$$k_1 = hf(t_n, y_n), \qquad k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right),$$

$$y_{n+1} = y_n + k_2 = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n)\right).$$

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$$

FIGURE 2.4 – Tableau de Butcher de la méthode RK2

Nous remarquons que la méthode RK2 est une composition de la méthode d'Euler. En effet, le principe de cette méthode est en premier lieu d'évaluer la dérivée au milieu de l'intervalle d'intégration de longueur $h = t_{n+1} - t_n$, *i.e.*

$$y'_{n+\frac{1}{2}} = f\left(t_n + \frac{h}{2}, y_{n+\frac{1}{2}}\right)$$

Pour évaluer $y_{n+\frac{1}{2}}$, nous utilisons la méthode d'Euler, *i.e.* :

$$y_{n+\frac{1}{2}} = y_n + \frac{h}{2} f(t_n, y_n)$$

ce qui permet d'obtenir (en repartant de l'équation différentielle) :

$$y'_{n+\frac{1}{2}} = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} f(t_n, y_n)\right).$$

À partir de cette estimation de la valeur de la dérivée au « point-milieu » $t_n + \frac{h}{2}$, nous approchons la valeur de y_{n+1} de la manière suivante :

$$y_{n+1} = y_n + h y'_{n+\frac{1}{2}}$$

ce qui permet de conclure sur l'expression de la méthode RK2, *i.e.* :

$$y_n + h f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} f(t_n, y_n)\right).$$

an a

- **Méthode de Runge-Kutta 4 (RK4)**

Au prix d'une implémentation plus coûteuse, la méthode de Runge-Kutta d'ordre 4 permet d'obtenir un degré supplémentaire de précision. Il s'agit de calculer une moyenne pondérée de l'évaluation de la dérivée (*via* les tangentes) au cours d'un pas d'intégration. Le tableau de Butcher de la méthode RK4 est décrit par la figure 2.2(a) et la relation de récurrence associée est :

$$k_1 = f(t_n, y_n), \qquad k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1\right),$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2\right), \qquad k_4 = f\left(t_n + h, y_n + h k_3\right),$$

$$y_{n+1} = y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).$$

2.3.4 Concept de stabilité(s)

En analyse numérique, il y a diverses manières d'interpréter le concept dit de *stabilité*. Il faut tout d'abord faire la distinction entre la notion de stabilité pour un système dynamique et la notion de stabilité pour une méthode numérique. Du point de vue des systèmes dynamiques, il s'agira généralement de la stabilité du système autour d'un point d'équilibre, souvent dénommée stabilité au sens de Lyapounov [138].

Du point de vue des algorithmes ou méthodes numériques, la stabilité est la propriété généralement requise pour assurer le « bon comportement » du système vis-à-vis des erreurs. De manière générale (et très informelle), une méthode est dite stable si elle permet de ne pas trop amplifier d'éventuelles erreurs (de méthode, d'arrondi, etc). La définition plus formelle dépend du contexte. Dans cette section, nous présentons brièvement deux définitions de la notion de stabilité pour les méthodes de Runge-Kutta explicites. Sont rencontrées dans la littérature de nombreuses autres caractérisations de la stabilité [45], comme la *A-stabilité* dans le cas où le système dynamique est raide.

0-stabilité

Parmi les définitions du concept de stabilité pour les méthodes de Runge-Kutta, la 0-*stabilité* [59, définition 2] est la plus fréquemment utilisée. Pour une petite erreur initiale et de petites erreurs « locales » à chaque itération de la méthode, la 0-stabilité assure que l'accumulation des erreurs est raisonnable. Plus précisément, considérons une méthode de Runge-Kutta, qui, appliquée à un système donné, est caractérisée par une relation de récurrence de la forme $y_{n+1} = y_n + h\Phi(t_n, y_n, h)$. La méthode est dite 0-stable pour le système sur l'intervalle d'intégration $[0; t_N]$ s'il existe $S \in \mathbb{R}_*^+$ tel que pour toutes suites $(y_n)_{n\in\mathbb{N}}$ et $(\widetilde{y_n})_{n\in\mathbb{N}}$ vérifiant :

$$\forall n < N, y_{n+1} = y_n + h\Phi(t_n, y_n, h), \qquad \widetilde{y_{n+1}} = \widetilde{y_n} + h\Phi(t_n, \widetilde{y_n}, h) + \varepsilon_n.$$

avec $(\varepsilon_n)_{n\in\mathbb{N}}$ la suite des erreurs locales, alors :

$$\forall n < N, |\widetilde{y_n} - y_n| \leqslant S\left(|\widetilde{y_0} - y_0| + \sum_{i=0}^{N} \varepsilon_i\right).$$

La constante S est appelée constante de stabilité de la méthode. La 0-stabilité est souvent liée aux notions de consistance et de convergence des méthodes numériques, la convergence étant assurée par la conjonction de la consistance et de la 0-stabilité. Ce critère de stabilité semble être un bon candidat pour borner l'accumulation des erreurs d'arrondi au cours des itérations. La 0-stabilité est par ailleurs très générale, s'appliquant à toute méthode numérique, quel que soit le système dynamique sur lequel elle s'applique. Cependant, les bornes d'erreurs obtenues sont pessimistes dans le cas général. En effet, si Φ est k-lipschitzienne, nous pouvons exhiber $S = e^{k\times t_N}$ [59], constante pouvant être très grande.

Stabilité linéaire

La *stabilité linéaire* [129] est un concept propre à l'application de méthodes de Runge-Kutta à des systèmes linéaires de la forme $y' = Ay$ (avec y, y' des vecteurs et A une matrice carrée). L'application d'une méthode de Runge-Kutta explicite à un système linéaire est caractérisée par une relation de récurrence de la forme $y_{n+1} = R(h, A)y_n$ (h étant le pas d'intégration de la méthode) où $R(h, A)$ est une matrice carrée. Par exemple, dans le cas de la méthode d'Euler, $R(h, A) = I + hA$. La stabilité linéaire s'exprime comme une propriété de la matrice $R(h, A)$.

Dans le cas unidimensionnel où A est remplacée par $\lambda \in \mathbb{R}$, la condition de stabilité est $|R(h, \lambda)| \leqslant 1$. Cette condition permet de déduire un intervalle de valeurs pour $h\lambda$ et donc de raffiner l'étude des erreurs d'arrondi.

Dans le cas multidimensionnel, l'idée générale est de décomposer y_n sur une base de vecteurs propres et d'imposer la condition de stabilité $|R(h, \lambda_i)| \leqslant 1$ sur chacune des valeurs propres λ_i de la matrice $R(h, A)$.

La stabilité linéaire est l'une des propriétés que nous utilisons pour raffiner les bornes d'erreurs d'arrondi des méthodes de Runge-Kutta (voir section 5.1.1).

Chapitre 3

État de l'art

Les résultats présentés dans ce manuscrit se situent à l'intersection de l'étude des erreurs d'arrondi de méthodes numériques et de la vérification formelle de résultats d'analyse numérique et fonctionnelle. Bien que spécialisés, ces domaines ont donné lieu à un éventail très varié de travaux de recherche. Ce chapitre propose un état de l'art non exhaustif de ces travaux et regroupe certaines contributions pertinentes au regard du sujet de ma thèse. La section 3.1 est vouée aux travaux étudiant les erreurs d'arrondi de méthodes numériques tandis que la section 3.2 présente quelques travaux de formalisation d'analyse numérique et fonctionnelle.

3.1 Erreurs d'arrondi de méthodes numériques

Bien que les mathématiciens soient conscients des problèmes causés par les erreurs d'arrondi, celles-ci sont généralement considérées comme négligeables par rapport aux erreurs de méthode des schémas numériques. Les erreurs d'arrondi sont par ailleurs peu étudiées à cause du manque d'outils ou méthodes efficaces dont dispose la communauté d'analyse numérique pour les borner. En 1937, bien avant l'apparition de l'arithmétique des ordinateurs, Brouwer [43] a utilisé une approche statistique pour estimer la propagation des erreurs d'arrondi associées à une méthode numérique d'intégration d'équations décrivant des mouvements planétaires. Cette contribution a creusé le sillage de travaux appliquant le même type d'approche pour diverses méthodes numériques (voir section 3.1.1). Une tendance qui s'est développée pour contourner la problématique des erreurs d'arrondi est de faire appel à l'argument de Brouwer [43] sans vérifier qu'il s'applique au problème considéré. La citation qui suit en témoigne [175] :

> *When stable difference equations are used, the rounding errors are not amplified as time goes on ; they merely accumulate roughly in proportion to the square root of the number of steps in the calculation.*

Or, il n'y a pas d'argument rigoureux qui permette de confirmer la validité de cette affirmation dans le cas général.

3.1.1 Approches statistiques et loi de Brouwer

Dans certains domaines de recherche, les temps d'intégration numérique peuvent s'avérer extrêmement longs. L'astrophysique est un exemple typique [21], les phénomènes étudiés (mouvement de planètes, évolution de galaxies, etc) évoluant sur des millions voire des milliards d'années. L'accumulation des erreurs d'arrondi sur de très grands intervalles d'intégration pouvant devenir problématique, elle a fait l'objet de nombreux travaux de recherche dans la communauté astronomique, l'approche adoptée étant le plus souvent statistique.

La contribution fondatrice est due à Brouwer [43], qui s'est intéressé aux erreurs d'arrondi dès 1937. Pour une équation donnée, décrivant l'évolution du mouvement de planètes, Brouwer observe une accumulation assez lente des erreurs d'arrondi, qu'il corrèle à des considérations statistiques. Il suppose que l'erreur d'arrondi commise au cours d'un pas d'itération est une variable aléatoire dont la moyenne est 0, *i.e.* que les signes des erreurs alternent au cours de l'intégration numérique. Il suppose également que la variance de cette variable aléatoire est un multiple de la racine carrée de l'unité d'arrondi. Lorsque la quantité étudiée est sujette à un phénomène de préservation (système dit hamiltonien, *e.g.* l'énergie), il en déduit que les erreurs d'arrondi peuvent être assimilées à des marches aléatoires et s'accumulent proportionnellement à la racine carrée du temps d'intégration (ou alternativement à la racine carrée du nombre d'itérations). La terminologie de « loi de Brouwer » est communément utilisée par la communauté des astronomes [91] et des numériciens [97]. Henrici a proposé une étude plus détaillée de la loi de Brouwer [105].

Quinlan [170] s'est intéressé à l'intégration numérique de l'orbite des planètes du système solaire sur une période de temps dont l'ordre de grandeur est l'âge du système solaire. Il a pour cela proposé l'utilisation de méthodes multi-pas appartenant à la classe des méthodes de Störmer [96]. Les expérimentations de Quinlan confirment l'hypothèse de Brouwer [43]. L'accumulation des erreurs est lente, mais cela ne lui semble cependant pas suffisant pour garantir la précision des résultats sur toute la durée d'intégration. L'auteur a donc proposé plusieurs techniques permettant de réduire significativement les erreurs d'arrondi, *e.g.* l'utilisation de rationnels ou de nombres flottants en précision étendue pour stocker les valeurs des résultats intermédiaires.

Grazier [91] s'est également intéressé aux mouvements des astres dans des systèmes à plusieurs corps comme le système solaire. Il a proposé l'intégration numérique d'équations différentielles de Newton *via* des méthodes multi-pas de Störmer. Comme Quinlan [170], il a observé une accumulation très importante d'erreurs, aussi bien de méthode que d'arrondi. Il a adapté l'implémentation des méthodes de Störmer utilisée pour permettre d'obtenir une erreur de méthode plus petite que l'unité d'arrondi du format flottant. Il a montré que cette implémentation permettait d'obtenir une accumulation des erreurs d'arrondi de l'ordre de la racine carrée du temps d'intégration, confirmant ainsi l'hypothèse de Brouwer [43] dans ce cas particulier.

Les travaux de Earn [63] et Skeel [188] ont proposé des techniques qui modifient l'implémentation des méthodes numériques et permettent d'éliminer certaines erreurs d'ar-

rondi. L'idée générale est de reformuler le système dynamique comme une « fonction-treillis » (de l'anglais *map lattice*) [121], une technique utilisée en théorie du chaos pour modéliser des systèmes non-linéaires. Skeel [188] a proposé une fonction d'arrondi dépendant du problème permettant de garantir la préservation d'énergie lors de l'utilisation des méthodes numériques malgré les erreurs d'arrondi. Earn [63] et Skeel [188] n'ont pas étudié précisément la propagation des erreurs mais ont donné une estimation statistique du ratio entre les erreurs de méthode et d'arrondi.

Fukushima [81] s'est intéressé aux techniques d'extrapolation, *e.g.* l'extrapolation de Richardson [174], qui permettent d'accélérer la convergence de méthodes numériques. L'auteur a plus particulièrement étudié les erreurs d'arrondi liées à l'utilisation de techniques d'extrapolation pour intégrer des équations orbitales, particulièrement sensibles aux problèmes de précision numérique. L'approche proposée pour réduire l'influence des erreurs d'arrondi est l'utilisation de sommations compensées (voir [155, §6]).

La plupart des travaux susmentionnés sont voués à l'étude des méthodes d'intégration multi-pas (voir section 2.3.2). Les auteurs ont généralement observé que l'implémentation de ces méthodes vérifiait l'hypothèse de Brouwer [43]. Hairer *et al.* [97] ont étudié l'accumulation des erreurs d'arrondi dans l'implémentation de méthodes à un pas, à savoir les méthodes de Runge-Kutta implicites. Au cours de l'intégration numérique d'un système hamiltonien, les auteurs ont observé une croissance linéaire des erreurs d'arrondi, observation qui vient infirmer la loi de Brouwer [43]. Ils ont émis plusieurs hypothèses qui permettent d'expliquer ce phénomène inattendu. Parmi les possibles explications, ils mentionnent le fait que les coefficients du tableau de Butcher des méthodes ne sont pas nécessairement représentables dans le format flottant. Ce problème serait responsable de l'accumulation systématique d'erreurs de même signe, qui ne se compensent pas au fil des itérations. Pour y remédier, Hairer *et al.* ont proposé de nouvelles implémentations qui utilisent des sommations compensées et font explicitement apparaître le terme d'erreur lié à chaque coefficient de la méthode. Ils ont utilisé des approches statistiques qui semblent confirmer la compatibilité de ces nouvelles implémentations avec la loi de Brouwer.

Antoñana *et al.* [5] ont raffiné les travaux de Hairer *et al.* [97] : leur principale contribution est l'utilisation d'une seconde implémentation de la même méthode numérique dans un format flottant de précision inférieure, qui permet de donner une estimation empirique de l'accumulation des erreurs d'arrondi.

D'autres travaux ont adopté des approches très différentes. C'est par exemple le cas de Sterne [196] au début des années 1950, qui a présenté une méthodologie générique permettant d'étudier la propagation des erreurs d'arrondi et de méthode au cours de la résolution de systèmes linéaires multidimensionnels. La méthodologie développée est fondée sur une analyse de sensibilité, qui consiste à considérer des facteurs d'incertitude sur la sortie d'un algorithme pour en déduire les facteurs d'incertitude sur les entrées. Sterne [196] s'est plus particulièrement intéressé à l'influence de la valeur de la condition initiale du système à résoudre. L'approche adoptée repose sur des considérations statistiques, même si elle ne repose pas sur l'hypothèse d'une distribution aléatoire des erreurs au cours des itérations.

3.1.2 Intégration numérique validée

L' *intégration numérique validée* constitue une autre solution permettant de se prémunir des erreurs liées à la résolution numérique d'équations différentielles. Cette technique consiste à implémenter les méthodes numériques en arithmétique d'intervalles afin de contrôler l'erreur totale, *i.e.* la somme des erreurs de méthode et d'arrondi.

Depuis les premiers travaux de Moore en 1966 [154], les séries de Taylor constituent la technique la plus répandue dans les travaux d'intégration numérique validée. En 1998, Berz et Makino [20] ont proposé l'intégration numérique de systèmes physiques simples, *e.g.* le mouvement d'une particule chargée en présence d'un aimant, *via* l'implémentation de séries de Taylor en arithmétique d'intervalles. Un an plus tard, les principes fondamentaux d'intégration numérique validée à base de séries de Taylor ont été détaillés par Nedialkov *et al.* [158], dont l'article sert aujourd'hui encore de référence à de nombreuses contributions scientifiques. Les travaux de Revol *et al.* [173] ont la particularité d'étudier plus finement la propagation des erreurs d'arrondi pour réduire la taille des intervalles, initialement trop pessimistes. L'implémentation proposée mélange l'arithmétique d'intervalles et l'arithmétique à virgule flottante. Des transformations exactes [155] sont utilisées sur la partie flottante du code afin de calculer les erreurs d'arrondi et de les ajouter dans l'intervalle correspondant aux erreurs de méthode.

Il existe également un ensemble de travaux plus récents proposant l'implémentation de méthodes de Runge-Kutta validées. Les travaux de Bouissou *et al.* [40, 39] se sont par exemple portés sur le calcul de bornes d'erreurs garanties pour l'implémentation de méthodes de Runge-Kutta. Alexandre dit Sandretto et Chapoutot [183] ont proposé une extension des travaux de Bouissou et Martel [40] à d'autres méthodes de Runge-Kutta, notamment les méthodes de Runge-Kutta implicites.

Les implémentations proposées par ces travaux produisent des résultats d'intégration garantis mais ne permettent pas d'analyser finement les erreurs d'arrondi, qui se confondent avec les erreurs de méthode.

3.1.3 Étude de la propagation des erreurs

Études expérimentales

Les approches statistiques partent du principe que les erreurs d'arrondi qui s'accumulent au cours des itérations d'une méthode numérique ont un signe qui alterne et que les erreurs sont aléatoirement distribuées. C'est le postulat fait par Rachemacher [171] en 1948 dans le cas de la méthode de Heun [59, p. 239] appliquée à des problèmes avec condition initiale de dimension 2. Un an plus tard, Huskey [111] a exhibé un contre-exemple assez simple invalidant le raisonnement de Rademacher. Huskey a utilisé la machine ENIAC [1], l'un des premiers ordinateurs entièrement électroniques, pour implémenter la résolution numérique du système suivant [2] *via* la méthode de Heun :

1. `http://eniacprogrammers.org/`
2. L'inconnue de cette équation est un vecteur de 2 fonctions, à savoir le sinus et le cosinus.

$$y'(t) = x(t), \qquad x'(t) = y(t), \qquad y(0) = 0, \qquad x(0) = 1.$$

Sur cet exemple, Huskey [111] a exhibé, au cours des itérations, des erreurs d'arrondi locales de signe constant et de taille comparable, invalidant ainsi l'argument de distribution aléatoire des erreurs. L'erreur globale exhibée est largement plus importante que ce que pouvait prévoir l'approche de Rademacher [171].

Études mathématiques à gros grain

Plusieurs auteurs ont proposé de borner l'accumulation des erreurs d'arrondi de méthodes numériques *via* des raisonnements mathématiques rigoureux et génériques. Nous utilisons la terminologie d'approche à « gros grain » pour qualifier les démonstrations utilisant les propriétés mathématiques des méthodes, mais pas les propriétés fines de l'arithmétique à virgule flottante.

Dans des travaux datant de 1970, Fehlberg [74] a évalué la propagation de l'erreur totale, *i.e.* la somme de l'erreur de méthode et de l'erreur d'arrondi, commise en utilisant des méthodes de Runge-Kutta explicites pour la résolution de systèmes de la forme :

$$x'(t) = f(t, x, y), \qquad y'(t) = g(t, x, y).$$

Fehlberg [74] a obtenu une formule explicite évaluant l'erreur totale, formule qui dépend des dérivées partielles de f et g, des erreurs de méthodes locales aux itérations et d'une estimation de l'erreur d'arrondi commise lors de l'évaluation de f et g. Ces travaux font cependant appel à des hypothèses simplificatrices. En particulier, Fehlberg utilise des développements en séries de Taylor pour la formule d'erreur totale, l'expression obtenue faisant apparaître des puissances du pas d'intégration h. L'auteur néglige les termes d'ordre supérieurs (en h^2, h^3, etc) pour des raisons de lisibilité.

Plus récemment, Spijker [192] a présenté une approche générique permettant de borner l'accumulation des erreurs d'arrondi dans la résolution numérique de systèmes non-linéaires avec condition initiale. Considérons un système multidimensionnel de la forme $y' = f(t, y)$ avec $y \in \mathbb{R}^n$. La méthodologie adoptée utilise la constante de Lipschitz associée à f pour borner les erreurs d'arrondi. Demailly [59] a proposé une approche similaire : considérons une méthode numérique à un pas constant dont l'application à un système donné définit la relation suivante :

$$y_{n+1} = y_n + h\Phi(t_n, y_n, h).$$

Une condition suffisante de 0-stabilité (voir section 2.3.4) pour la méthode est le caractère k-lipschitzien de Φ en y_n, qui permet d'obtenir la constante de 0-stabilité e^{kT} avec T la taille de l'intervalle d'intégration. Dans le cas où Φ est k-lipschitzienne, Demailly en déduit une borne sur l'erreur d'arrondi accumulée qui dépend de e^{kT} et des caractéristiques du format flottant. Cette borne, bien que très générique, s'avère pessimiste dans la majorité des cas.

Kalinina [120] a étudié l'erreur totale commise au cours de l'implémentation de méthodes numériques à un pas variable, fondées sur la méthode d'Euler et utilisées pour la résolution de systèmes non-linéaires de la forme $y' = f(t, y)$. Le but principal de ces travaux est d'exhiber le pas d'intégration optimal qui minimise l'erreur totale (somme de l'erreur de méthode et de l'erreur d'arrondi) commise au cours d'une itération du schéma numérique. Les résultats exhibés montrent que l'erreur totale est minimisée lorsque le pas d'intégration permet d'obtenir une erreur de méthode dont la magnitude est de l'ordre de celle de l'erreur d'arrondi. Comme dans les travaux de Fehlberg [74], l'erreur d'arrondi commise lors de l'évaluation de f est supposée connue.

Les travaux présentés dans cette section permettent d'obtenir des bornes généralement valides mais assez grossières, qui n'utilisent pas les propriétés fines de l'arithmétique à virgule flottante de la norme IEEE-754 [113]. En particulier, les dépassements de capacité (voir section 2.1) ne sont pas pris en compte.

Études mathématiques à grain fin

Dans les communautés du calcul formel et de l'arithmétique des ordinateurs, des analyses plus fines ont été réalisées. Nous utiliserons la terminologie d'approche à « grain fin » pour qualifier les démonstrations décomposant l'analyse d'erreurs jusqu'à l'échelle des opérations flottantes élémentaires. Ces démonstrations utilisent généralement les propriétés fines du format d'arithmétique utilisé.

Fousse [79, 78, 77] a proposé une analyse d'erreurs pour l'implémentation en précision arbitraire de méthodes numériques de calcul d'intégrales. Les méthodes étudiées sont les formules de quadrature de Newton-Cotes [79] et de Gauss-Legendre [78]. Dans les deux cas, l'auteur a borné à la fois l'erreur de méthode et l'erreur d'arrondi, afin de pouvoir les comparer et d'obtenir une borne sur l'erreur totale par rapport à la solution exacte. Les bornes sur les erreurs d'arrondi sont génériques et paramétrées par la précision p de calcul et certaines données de la méthode, *e.g.* le nombre n de pas d'intégration ou des résultats de calculs intermédiaires de l'implémentation. Les algorithmes étudiés ont fait l'objet d'une implémentation utilisant MPFR [80], une bibliothèque de calcul en arrondi correct avec des nombres flottants multi-précision.

Fousse a d'abord étudié la méthode de Newton-Cotes [79],[77, §3] : cette méthode fait intervenir un ensemble de coefficients qui peuvent être précalculés en amont de l'application de la méthode. L'auteur a donc distingué l'analyse d'erreurs sur les coefficients, dont il a prouvé qu'ils étaient calculés de manière exacte, et l'analyse d'erreurs pour la méthode elle-même. Il a également proposé une expérimentation numérique pour une fonction dont la valeur exacte de l'intégrale est connue, et a constaté que l'erreur d'arrondi dominait l'erreur de méthode au-delà d'un certain nombre de pas d'intégration. L'expérimentation a permis d'éprouver la borne obtenue, qui s'avère relativement fine.

La seconde méthode étudiée est la méthode de Gauss-Legendre [78], qui repose sur des fondements mathématiques plus complexes. L'approche adoptée par l'auteur est similaire à celle utilisée dans le cas de la méthode de Newton-Cotes. En particulier,

Fousse s'est d'abord intéressé à des coefficients pouvant être précalculés, puis a borné l'erreur totale associée à l'implémentation de la méthode. Dans ce cas, contrairement à la méthode de Newton-Cotes, des expérimentations numériques ont montré que l'augmentation de l'ordre de la méthode ne débouchait pas sur une augmentation catastrophique des erreurs d'arrondi et, par conséquent, ne diminuait pas la précision numérique.

Il existe de nombreuses similarités entre les méthodes de quadrature et les méthodes numériques de résolution d'équations différentielles, qui reposent sur un socle commun de fondements mathématiques. Une différence majeure est le fait que l'expression de la fonction à intégrer par une méthode de quadrature est généralement connue, contrairement à l'inconnue d'une équation différentielle. Par exemple, dans [78], Fousse a supposé connue une borne sur certaines dérivées de la fonction à intégrer. Nous adoptons, dans la partie II de ce manuscrit, une approche assez similaire à celle de Fousse, à savoir une analyse d'erreurs à grains fins. Néanmoins, nous ne proposons pas de borner l'erreur de méthode, ce qui pourrait constituer une perspective de travail intéressante (voir chapitre 9). En revanche, contrairement à ce que nous proposons, les preuves de Fousse n'incluent pas la gestion des dépassements de capacité inférieurs. L'algorithme est en effet prévu pour une implémentation en multi-précision *via* MPFR, dont la plage d'exposants est très grande.

3.2 Preuves formelles en analyse numérique

Les méthodes d'analyse numérique sont fréquemment utilisées dans le cadre d'applications industrielles critiques, *e.g.* en aéronautique [190], en médecine [198] ou dans le secteur de l'automobile [6, 127], qui suscitent un fort intérêt pour des travaux de certification. Avec l'émergence de formalisations d'analyse réelle dans divers assistants de preuves (voir la revue de la littérature proposée par Boldo, Lelay et Melquiond [36]), la vérification formelle de techniques d'analyse numérique est devenue possible.

Dans cette section, nous présentons des travaux de formalisations liés au domaine de l'analyse numérique, *i.e.* un ensemble de techniques calculatoires permettant d'obtenir l'approximation de la solution d'un problème mathématique difficile, comme la résolution d'équations différentielles. La section 3.2.1 présente des travaux liés à la preuve de propriétés de méthodes numériques. En section 3.2.2, nous décrivons des travaux de formalisation d'analyses d'erreurs d'arrondi liées à l'implémentation de méthodes numériques en arithmétique à virgule flottante. La section 3.2.3 est vouée aux travaux de formalisation de fondements sous-jacents aux méthodes de résolution d'équations aux dérivées partielles, incluant l'analyse fonctionnelle et les espaces de dimension finie.

3.2.1 Vérification de méthodes numériques

Différentiation automatique

L'une des premières utilisations d'un assistant de preuves pour des problèmes d'analyse numérique remonte aux travaux de Mayero au début des années 2000 [148, §5], [149],

voués à la vérification formelle de méthodes de différentiation automatique en Coq. La différentiation automatique consiste à calculer numériquement une approximation de la dérivée d'une fonction en un point. Mayero [149] a vérifié l'algorithme de différentiation de l'outil O∂yssée [73], notamment utilisé pour le calcul de gradients.

Lelay et Melquiond [135] ont implémenté une tactique Coq réflexive [17, §17] permettant d'automatiser ou simplifier les preuves de dérivabilité et le calcul de dérivées de fonctions. Ce développement est basé sur Coquelicot [35, 134].

Calcul numérique d'intégrales

Mahboubi, Melquiond et Sibut-Pinote [141, 142, 187] ont proposé une méthode efficace pour calculer et borner des intégrales définies dans Coq. La technique développée repose essentiellement sur l'utilisation de l'arithmétique d'intervalles et d'approximations rigoureuses de polynômes. Les auteurs se sont d'abord intéressés aux intégrales (au sens de Riemann) dites *propres* [141], *i.e.* de la forme $\int_a^b f(t)dt$, telles que $a, b \in \mathbb{R}$ et telles que la fonction f soit définie et bornée sur $[a, b]$ et discontinue en un nombre fini de points. Ils ont ensuite étendu leur étude au cas des intégrales impropres [142] dont les bornes peuvent être infinies. Notons que ces travaux ont été intégrés à l'outil CoqInterval [150], dont ils constituent désormais une extension.

Analyse numérique des équations différentielles ordinaires

Immler et Hölz ont formalisé le concept d'équation différentielle ordinaire en Isabelle/HOL [116]. Ils ont plus précisément spécifié la notion de problème avec condition initiale et ont démontré formellement le théorème de Cauchy-Lipschitz [3] assurant l'existence d'une unique solution au problème sous certaines conditions [116, §4]. Les auteurs ont ensuite formalisé le concept générique de méthode à un pas pour la résolution numérique d'un problème avec condition initiale [116, §5]. Ils ont notamment défini les notions de consistance, de 0-stabilité et de convergence (voir [59] pour une définition précise de ces termes) pour les méthodes à un pas, puis ont démontré les liens qui existent entre ces trois notions. Les auteurs se sont intéressés au cas particulier de la méthode d'Euler [116, §6] et ont formellement prouvé la convergence de la méthode, même en présence de perturbations sur les calculs (*e.g.* des erreurs d'arrondi).

Makarov et Spitters [146] ont également proposé la formalisation d'une autre version du théorème de Cauchy-Lipschitz en Coq, reposant sur des fondements mathématiques très différents. Cela s'explique par le fait que la bibliothèque sous-jacente qui est utilisée soit la bibliothèque C-CoRN d'analyse réelle constructive [55].

Immler [114] a implémenté une méthode d'Euler garantie à base d'arithmétique affine dans Isabelle/HOL. La correction de la méthode est vérifiée, la démonstration de ce résultat reposant sur la formalisation des problèmes avec conditions initiales qu'ont proposé Immler et Hölz dans de précédents travaux [116, §4]. L'idée générale est très proche de celle adoptée dans les travaux d'intégration numérique garantie présentés en

3. Dans l'article [116], ainsi que dans [146], la terminologie *Picard-Lindelöf* désigne le nom utilisé par les anglophones pour le théorème de Cauchy-Lipschitz.

section 3.1.2 et plus particulièrement dans les travaux de Bouissou *et al.* [39]. La plus-value apportée par Immler par rapport à ces travaux est la garantie formelle que permet l'usage d'Isabelle/HOL. Immler et Traut [117] ont complété cette série de travaux en proposant une étude formelle du concept de flot d'équation différentielle. Le flot d'une équation différentielle ordinaire correspond à l'évolution de la solution en fonction de l'évolution de la condition initiale, *i.e.* la valeur de la fonction au début de l'intervalle d'intégration. Le socle mathématique de ces travaux (que nous présentons plus en détails en section 3.2.3) est proche de celui que nous développons dans le chapitre 8. L'ensemble des travaux d'Immler *et al.* [116, 114, 117] présentés ci-dessus servent de fondements à l'implémentation d'un solveur d'équations différentielles en Isabelle/HOL [115], qui a été utilisé pour démontrer les propriétés d'un attracteur de Lorenz [201].

Il existe également des travaux appliqués à la théorie du contrôle ou aux systèmes cyber-physiques et dont les fondements théoriques incluent la théorie des équations différentielles. Rouhling [176] a par exemple formalisé dans Coq la preuve de correction d'une fonction de contrôle pour un pendule inversé. Nous pouvons également citer les travaux de Bohrer *et al.* [22] autour de la vérification de propriétés de la logique dynamique différentielle implémentée dans des outils tels que KeYmaera X [167], permettant de raisonner sur des systèmes hybrides modélisés par des équations différentielles.

Analyse numérique des équations aux dérivées partielles

Boldo *et al.* [28] ont formalisé en Coq l'étude d'un schéma de résolution de l'équation des ondes, un cas particulier d'équation aux dérivées partielles, en dimension 1. Les auteurs se sont plus particulièrement intéressés à la propriété de convergence, et *a fortiori* à l'erreur de méthode, de ce schéma numérique. Ce travail va de pair avec les travaux de Boldo [23] sur l'étude des erreurs d'arrondi de ce même schéma numérique (que nous détaillons en section 3.2.2), ces deux contributions servant de base à la preuve formelle complète d'un programme C de résolution de l'équation des ondes [25]. Bien que ces travaux traitent les équations aux dérivées partielles, plus difficiles à appréhender que les équations différentielles ordinaires, notre approche est plus générique car elle traite une large classe de méthodes et systèmes, et non un exemple particulier d'équation.

Approximation des racines de fonctions

Pasça a formalisé une étude de la méthode de Newton en Coq [163, §3], [161, 162]. La méthode de Newton est un algorithme itératif permettant d'obtenir une valeur approchée des racines d'une fonction $f : \mathbb{R}^n \to \mathbb{R}^n$. À partir d'une approximation initiale x_0 de la racine, la méthode de Newton calcule successivement des approximations x_i qui sont de plus en plus proches de la solution exacte. Pasça a démontré formellement certains résultats de stabilité ou de convergence pour la méthode de Newton, notamment le théorème de Kantorovitch assurant la convergence de la méthode sous certaines conditions [161]. L'auteure [162] a proposé une variante du théorème de Kantorovitch en tenant compte d'un arrondi à chaque itération de la méthode de Newton, l'arrondi

étant entendu au sens très général comme une fonction produisant une perturbation.

Dans le cas des systèmes multidimensionnels, la méthode de Newton fait intervenir des vecteurs et des matrices, notamment jacobiennes. Pasça a proposé une formalisation de la norme matricielle $|||\cdot|||_\infty$ telle que pour toute matrice A, $|||A|||_\infty = \max_i \sum_j |A_{ij}|$ [161]. Cette formalisation est fondée sur les grands opérateurs de la bibliothèque Mathematical Components [18]. Dans la partie II de ce manuscrit, nous utilisons cette même norme $|||\cdot|||_\infty$ et nous nous inspirons des choix de formalisation de l'auteure.

3.2.2 Formalisations d'erreurs d'arrondi en analyse numérique

L'implémentation efficace d'algorithmes numériques requiert l'utilisation de structures de données en précision finie, qui induisent des erreurs d'arrondi. Dans cette section, nous nous intéressons à des travaux de formalisation qui se sont plus particulièrement penchés sur l'étude de ces erreurs d'arrondi.

Calcul d'opérateurs flottants et de fonctions élémentaires

L'usage d'assistants de preuves dans la démonstration de propriétés liées à l'arithmétique à virgule flottante a commencé à émerger dans les années 1990 avec une spécification en Z de la norme IEEE-754 proposée par Barrett [11] ainsi que des formalisations de la norme IEEE-854 [51] [4] en HOL et PVS, qui sont dues à Carreño et Miner [47].

En 1998, Russinoff [182] a vérifié formellement les propriétés d'instructions du processeur AMD-K7 [159] dans l'assistant de preuves ACL2. Les instructions auxquelles l'auteur s'intéresse sont la multiplication, la division et la racine carrée en arithmétique à virgule flottante. Russinoff a formalisé une spécification bas niveau de l'arithmétique à virgule flottante dans ACL2 et a proposé de nombreux résultats, incluant des analyses d'erreurs d'arrondi. La difficulté principale relevée par l'auteur est le fossé qui sépare l'implémentation matérielle très bas niveau des instructions flottantes et les preuves de correction associées, souvent très abstraites.

Dans des travaux similaires qui utilisent également ACL2, Sawada [184] a proposé une formalisation d'une analyse d'erreurs pour des algorithmes de division et de calcul de racine carrée utilisés sur le processeur IBM Power4 [199]. Ces algorithmes sont techniques et reposent sur des approximations à base de séries, *e.g.* des séries de Chebyshev dans le cas de la racine carrée. La démonstration de Sawada est une analyse « à grains fins » (voir section 3.1.3) puisqu'il décompose l'analyse d'erreurs sur chaque pas de l'algorithme. La borne obtenue est précise et dépend de l'*ulp* des entrées (voir section 2.1.2).

Harrison [102] a formalisé dans HOL-Light l'étude d'algorithmes approchant les fonctions trigonométriques *sinus* et *cosinus* par des développements en séries de Taylor. Le développement utilise une formalisation HOL de l'arithmétique à virgule flottante proposée par le même auteur [101]. Harrison a proposé une analyse à grains fins et un

4. La norme IEEE-854 [51] est indépendante de la base de calcul.

résultat final dépendant de l'*ulp* du paramètre des fonctions *sinus* et *cosinus*. Dans le même esprit, Harrison s'est également intéressé à la vérification formelle d'un algorithme calculant la fonction exponentielle en arithmétique à virgule flottante [100].

De Dinechin, Lauter et Melquiond [61] ont utilisé l'outil Gappa [57, 61] pour vérifier des propriétés fines d'algorithmes optimisés calculant certaines fonctions élémentaires. L'exemple de programme servant de fil rouge à l'article est extrait du code de la fonction *sin* de la bibliothèque CRlibm [56], qui propose l'implémentation en arrondi correct de certaines fonctions élémentaires.

Calcul de constantes mathématiques

Bertot *et al.* [19] ont étudié l'implémentation en arithmétique à virgule fixe de plusieurs algorithmes permettant de calculer le nombre π avec un grand nombre de décimales. Les méthodes étudiées sont variées : une première méthode repose sur la formule de Bailey–Borwein–Plouffe (BBP) [9] alors que les suivantes reposent sur des calculs de moyennes arithmético-géométriques. Les calculs sont réalisés dans l'assistant de preuves Coq, ce qui permet aux auteurs de garantir formellement les algorithmes et les erreurs d'arrondi qui leur sont associées.

Résolution numérique des équations aux dérivées partielles

En complément de l'erreur de méthode du schéma de résolution de l'équation des ondes (voir section 3.2.1, paragraphe 4), Boldo s'est intéressée aux erreurs d'arrondi accumulées lors de l'utilisation de ce même schéma numérique [23]. L'approche de Boldo consiste à développer une expression analytique de l'erreur d'arrondi globale en fonction des erreurs locales signées associées aux itérations précédentes. Contrairement à l'approche qui consiste à prouver par récurrence une borne sur la valeur absolue de l'erreur globale, l'approche de Boldo, empruntée aux principes d'arithmétique affine, a permis d'exhiber des compensations d'erreurs et d'obtenir une borne plus fine sur l'erreur finale.

Analyse matricielle

Roux [177] a proposé une analyse d'erreurs d'arrondi en Coq pour plusieurs algorithmes numériques impliquant des opérations matricielles, comme la décomposition de Cholesky ou la vérification d'invariants ellipsoïdaux utilisés par des contrôleurs linéaires. Cette formalisation repose sur une spécification de l'arithmétique à virgule flottante définie *via* un type `Record` et particulièrement adaptée à l'analyse d'erreurs. Cette spécification est par ailleurs satisfaite par le format *binary64* de Flocq [38] (voir section 2.2.4). L'un des résultats intermédiaires formellement démontrés par Roux est une borne sur l'erreur d'arrondi du produit scalaire de deux vecteurs.

Dans le chapitre 6 de ce manuscrit, nous démontrons en Coq un résultat très similaire qui ne réutilise pas ces précédents travaux. Notre approche diffère en effet légèrement de celle adoptée par Roux [177] car nous utilisons directement la formalisation des formats

de nombres flottants de la bibliothèque Flocq, sans passer par une abstraction dédiée de l'arithmétique à virgule flottante.

Martin-Dorel et Roux [147] ont développé une bibliothèque appelée ValidSDP, qui propose une tactique Coq réflexive pour la preuve d'inégalités sur des polynômes multivariés et se base directement sur les développements de Roux [177].

3.2.3 Analyse fonctionnelle et espaces de dimension finie

Analyse fonctionnelle

À notre connaissance, les travaux les plus anciens qui puissent être qualifiés de formalisations d'analyse fonctionnelle remontent à Popiołek en 1991 [168] qui formalise les notions d'espaces complets et d'espaces de Hilbert dans l'assistant de preuves Mizar. Cette spécification sert de base à d'autres travaux utilisant Mizar, *e.g.* la formalisation des opérateurs linéaires bornés par Shidama [186].

Pour les besoins de la formalisation de flots d'équations différentielles, Immler et Traut [117] ont formalisé le concept de fonction linéaire continue. Leur formalisation est sur ce point très proche de la nôtre (voir chapitre 7) : en particulier, ils ont caractérisé la continuité d'une fonction *via* la propriété $\exists K, \forall y, |f(y)| \leqslant K \cdot |y|$ et la norme d'opérateur comme $\max_{|y|\leqslant 1} |f(y)|$. Notre développement et le leur ont été menés en parallèle et de manière indépendante, mais reposent sur des fondements très similaires d'analyse et de topologie. Nous nous fondons en effet sur Coquelicot [35], dont les définitions topologiques (utilisant le formalisme des filtres) et la hiérarchie algébrique sont inspirés des développements de Hölzl, Immler et Huffman [109] en Isabelle/HOL, travaux sur lesquels repose la formalisation d'Immler et Traut [117].

Le théorème de Hahn-Banach [42] est un théorème de prolongement de formes linéaires continues qui fait partie des résultats classiques d'analyse fonctionnelle. Il existe plusieurs travaux de formalisation du théorème de Hahn-Banach.

En 1999, Bauer et Wenzel [15] ont proposé une formalisation du théorème de Hahn-Banach en Isabelle/HOL. Leur preuve formelle est présentée comme une preuve de concept de l'utilisation du langage Isar [205], qui permet de développer des formalisations mathématiques lisibles et très proches du langage mathématique usuel. La démonstration repose sur la formalisation de divers concepts d'analyse (fonctionnelle notamment), comme les espaces vectoriels, les formes linéaires d'un espace vectoriel ou la notion de norme d'opérateur pour une fonction, que nous avons également définie en Coq pour la démonstration du théorème de Lax–Milgram (voir chapitre 7).

Plus récemment, Kerjean et Mahboubi [122] ont présenté une preuve formelle du théorème de Hahn-Banach en Coq, utilisant la formalisation des espaces vectoriels de la bibliothèque Mathematical Components [144].

Narkawicz [156] a proposé une démonstration formelle du théorème de Riesz–Markov–Kakutani [5] dans l'assistant de preuves PVS. Ce résultat très général représente, dans un

5. à ne pas confondre avec le théorème de Riesz–Fréchet que nous démontrons dans le chapitre 8.

espace topologique de Hausdorff, les éléments du dual de l'espace des fonctions continues à support compact par une mesure *via* la notion d'intégrale. Dans le développement de Narkawicz, l'intégrale est considérée au sens de Riemann–Stieltjes, une généralisation de l'intégrale de Riemann. La démonstration formelle est conséquente : Narkawicz a par exemple démontré l'équivalence entre intégration de Riemann–Stieltjes et intégration de Darboux–Stieltjes. La démonstration utilise également un cas particulier du théorème de Hahn-Banach.

J'ai mené, en collaboration avec Spitters [71, 72], un travail de démonstrations constructives de programmes probabilistes en Coq *via* le formalisme de la théorie homotopique des types [13]. Nous avons partiellement démontré un cas particulier du théorème de Riesz–Markov–Kakutani pour la théorie des valuations dans un contexte dit de *topologie synthétique* [65, 14].

Espaces de dimension finie, espaces euclidiens

Dans la partie III de ce manuscrit, et plus précisément dans le chapitre 8, nous nous intéressons à la formalisation de sous-espaces de dimension finie d'espaces de Hilbert.

Harrison [103, 104] a proposé une formalisation des espaces euclidiens en HOL-Light, un espace euclidien étant un espace de Hilbert de dimension finie. L'auteur a caractérisé un espace euclidien par un type fonctionnel $A \to E$ avec A un type de cardinal fini n, *i.e.* un type à n éléments, et E un espace de Hilbert. Les exemples les plus intuitifs d'espaces euclidiens sont les espaces de la forme $\mathbb{R}^n$. Harrison propose la caractérisation de l'espace $\mathbb{R}^n$ par le type $A \to \mathbb{R}$ avec $A = Unit + \cdots + Unit$ (n fois).

Les travaux de Hölz, Immler et Huffman [109] en Isabelle/HOL incluent une formalisation des espaces euclidiens. Ces espaces sont définis, *via* le mécanisme des classes de types (*type classes* d'Isabelle) [95], en tant qu'espaces vectoriels munis d'un produit scalaire et d'une base finie de vecteurs, un élément de l'espace de dimension finie étant obtenu comme combinaison linéaire des vecteurs de la base.

En Coq, Brunel [44] a également formalisé les espaces euclidiens de la forme $\mathbb{R}^n$ comme produit cartésien par induction sur la dimension n. La technique utilisée pour munir $\mathbb{R}^n$ de la structure d'espace euclidien (produit scalaire et ses propriétés fondamentales) est le mécanisme des classes de types (*type classes* de Coq [191]). Cette définition sert de base à une formalisation de résultats d'analyse dans $\mathbb{C}$.

La bibliothèque Mathematical Components [144] inclut également une structure d'espaces vectoriels de dimension finie, qui est quant à elle définie en utilisant le mécanisme des structures canoniques [143].

Dans les travaux présentés ci-dessus [103, 104, 109, 44, 144], les espaces de dimension finie ne sont pas « par construction » des sous-espaces d'autres espaces plus grands. Il est en revanche possible de démontrer qu'un espace de dimension finie E_1 est inclus dans un espace E_2 de dimension supérieure, notamment lorsque la base associée à E_1 est une sous-famille de la base associée à E_2. Dans le chapitre 8, nous souhaitons définir des sous-espace de dimension finie d'espaces de Hilbert de dimension potentiellement infinie. À notre connaissance, il y a peu de travaux qui traitent ce genre de formalisations.

Nous pouvons citer les travaux de Afshar *et al.* [3] et de Mahmoud *et al.* [145] en Isabelle/HOL, qui ont défini des espaces vectoriels sur $\mathbb{C}$ de dimensions à la fois finies et infinies. Néanmoins, ces travaux ne traitent pas des espaces de Hilbert génériques.

3.3 Conclusion

Nous avons présenté une sélection de travaux traitant d'une part l'étude des erreurs d'arrondi dans l'implémentation des méthodes numériques et d'autre part la formalisation de méthodes numériques, en partant des fondements mathématiques sur lesquels elles s'appuient jusqu'à l'analyse de leur implémentation flottante.

Concernant l'étude des erreurs d'arrondi, nous avons pu identifier plusieurs approches de nature très différente.

Les travaux les plus anciens [43, 196, 91, 170, 63, 188, 81, 97, 5] (voir section 3.1.1), qui ont émergé dans la communauté des astrophysiciens, proposent d'améliorer les implémentations de méthodes numériques pour permettre de réduire l'influence des erreurs d'arrondi. L'approche adoptée pour vérifier les résultats est généralement statistique.

Une autre approche, l'intégration numérique validée, consiste à implémenter des méthodes numériques en arithmétique d'intervalles pour garantir le contrôle de l'erreur totale (somme de l'erreur de méthode et de l'erreur d'arrondi) [20, 158, 40, 39, 183, 173, 4] (voir section 3.1.2). Si ces approches sont rigoureuses, elles ne tiennent compte ni des propriétés mathématiques des méthodes, ni des propriétés fines de l'arithmétique à virgule flottante. Par ailleurs, elles ne permettent pas de distinguer les différents types d'erreurs, ni d'identifier la source des erreurs d'arrondi.

D'autres travaux utilisent une approche mathématique rigoureuse pour estimer et borner l'accumulation des erreurs d'arrondi de méthodes itératives [74, 192, 59, 120, 79, 78, 77]. La majeure partie de ces travaux [74, 192, 59, 120] adopte une approche à « gros grain » : les résultats exhibés sont très génériques mais ne prennent pas en compte les propriétés fines de l'arithmétique à virgule flottante. En revanche, les contributions de Fousse [79, 78, 77] visant à borner les erreurs d'arrondi de méthodes numériques de calcul d'intégrales adoptent une approche à « grains fins » en décomposant l'analyse d'erreurs jusqu'au niveau des opérations flottantes élémentaires.

En ce qui concerne l'utilisation des assistants de preuves dans le domaine de l'analyse numérique, nous avons identifié un nombre important de contributions récentes.

Une première série de travaux est dédiée à la vérification formelles de propriétés de méthodes numériques (voir section 3.2.1). Les méthodes numériques sont variées : nous avons identifié des travaux liés à la différentiation automatique [149, 135], au calcul numérique d'intégrales [141, 142, 187], à l'étude des équations différentielles ordinaires [116, 146, 114, 117, 176, 22] ou aux dérivées partielles [28, 25], l'approximation de racines de fonctions [161, 162] et l'étude de matrices-intervalles [164].

Une seconde série de travaux est dédiée à l'analyse formelle de l'implémentation flottante de méthodes numériques, et plus particulièrement aux erreurs d'arrondi (voir

section 3.2.2). Les travaux les plus anciens concernent le calcul de fonctions élémentaires [182, 184, 102] tandis que des contributions plus récentes sont vouées à l'implémentation de schémas numériques pour la résolution d'équations différentielles [23] ou à des problématiques d'analyse matricielle [177].

Enfin, la dernière série de travaux est vouée à la formalisation de résultats théoriques liés à l'analyse fonctionnelle [168, 186, 117, 15, 122, 156, 71] et aux espaces de dimension finie [103, 104, 109, 44, 144, 145, 3], qui font l'objet de la partie III du manuscrit.

Il faut noter que ces travaux de formalisation ont fait usage de divers assistants de preuves, nommément Coq [149, 135, 141, 142, 187, 146, 28, 25, 161, 162, 164, 23, 177, 122, 71, 44, 144], HOL-Light [102, 103, 104], Isabelle/HOL [116, 114, 117, 22, 15, 109, 145, 3], Mizar [168, 186], PVS [156] et ACL2 [182, 184].

Première partie

Moyenne de nombres flottants décimaux

Chapitre 4

Arrondi correct de la moyenne de deux nombres flottants décimaux

La norme IEEE 754 définit les formats de représentation des nombres à virgule flottante ainsi qu'un ensemble d'opérations sur ces nombres [112] (voir chapitre 2). Dans la version initiale datant de 1985, la norme ne spécifie que des formats de représentation de nombres flottants binaires. Or, comme le fait remarquer Cowlishaw à l'aube des années 2000 [54], les données et règles de calcul utilisées dans les applications commerciales et financières sont par nature liées au système de calcul décimal.

En 2008, la norme a subi une révision majeure incluant des formats de représentation pour les nombres flottants décimaux [113]. Cela va de pair avec la production d'unités matérielles dédiées au calcul en base 10, comme les processeurs IBM POWER6 [131] et z9 [62]. Ces avancées conduisent à l'émergence de nouveaux axes de recherche portant sur l'arithmétique décimale.

Malheureusement, il n'est ni automatique ni trivial de transposer aux nombres flottants décimaux des propriétés valides pour des nombres flottants binaires. Par exemple, un algorithme fournissant l'arrondi correct d'une opération en base 2 ne fournit pas nécessairement l'arrondi correct en base 10 [132]. Les programmes en apparence les plus simples n'échappent pas à la règle, comme la moyenne de deux nombres flottants :

$$\circ^{\tau}\left(\frac{x+y}{2}\right),$$

$\circ$ étant l'arrondi au plus proche, avec règle de bris d'égalité τ quelconque.

Dans ce chapitre, nous commençons par présenter les critères (tels qu'énoncés par Sterbenz [195]) que devrait satisfaire tout programme fournissant la moyenne de deux nombres flottants puis présentons quelques algorithmes issus de précédents travaux et satisfaisant ces critères pour des formats binaires (section 4.1). Ensuite, nous montrons que les implémentations les plus intuitives ne fournissent pas l'arrondi correct lors du passage en base 10 (section 4.2) puis exhibons un algorithme plus élaboré calculant l'arrondi correct de la moyenne de deux nombres flottants décimaux (section 4.3). Nous proposons une démonstration de la propriété d'arrondi correct de l'algorithme : nous

la présentons tout d'abord sans tenir compte des comportements exceptionnels pour des questions de lisibilité (section 4.3.1) et décrivons la preuve formelle associée (section 4.3.2) dans le format `FLX` de Coq. Enfin, nous montrons comment Coq a aidé à la généralisation des démonstrations en les rendant notamment valides dans le format `FLT`, *i.e.* en tenant compte d'éventuels dépassements graduels inférieurs (section 4.3.3). Les résultats de ce chapitre ont fait l'objet d'un article avec Boldo et Tourneur [31].

4.1 Calcul de moyenne en base 2

4.1.1 Critères de Sterbenz

Les questions concernant le calcul de la moyenne de nombres à virgule flottante ont commencé à émerger vers le début des années 1970. Sterbenz est l'un des premiers à aborder ce sujet [195]. Son but n'est pas d'exhiber un algorithme canonique de calcul de moyenne qui soit à la fois efficace et précis, mais plutôt de donner un ensemble de critères que doit réunir un tel algorithme. L'algorithme doit par exemple être symétrique et suffisamment précis. Sterbenz propose par ailleurs de bonnes pratiques pour prévenir les dépassements de capacité supérieurs et pour gérer les dépassements de capacité inférieurs. Ces pratiques s'inscrivent dans l'optique d'une programmation minutieuse allant de pair avec une prise en compte des détails inhérents au format de représentation des nombres [1].

Sur la base de ces critères, quelques implémentations potentielles sont présentées :

- $(x \oplus y) \oslash 2$, qui est très précise et fournit l'arrondi correct en base 2 mais peut provoquer un dépassement de capacité supérieur lorsque x et y sont de même signe (pour rappel, $\oplus$, $\otimes$, etc, désignent les opérations arrondies (voir section 2.1.3)).
- $(x \oslash 2) \oplus (y \oslash 2)$, qui est également précise mais ne fournit pas nécessairement l'arrondi correct en cas de dépassement de capacité inférieur.
- $x \oplus ((y \ominus x) \oslash 2)$, qui est moins précise que le premier algorithme, mais ne provoque pas de dépassement parasite de capacité supérieur lorsque x et y sont de même signe.

Nous ne présentons ci-dessus que quelques éléments d'analyse de ces algorithmes ; l'étude menée par Sterbenz [195] est plus exhaustive et confronte les trois implémentations à l'ensemble des critères qu'il énonce.

4.1.2 Algorithmes de calcul de moyenne en base 2

Nous remarquons qu'aucune des implémentations présentées en section 4.1.1 n'est exempte de défaut ou de compromis. Il s'avère donc nécessaire de combiner ces formules à des tests bien choisis pour obtenir des programmes précis, voire corrects, sans faire d'impasse sur l'efficacité. Dans cette section, nous présentons des travaux antérieurs visant à calculer l'arrondi correct de la moyenne de deux nombres flottants binaires.

1. La contribution de Sterbenz [195, §8] est d'ailleurs intitulée « carefully written programs ».

Boldo propose deux algorithmes formellement prouvés [24]. Le premier algorithme est précis et respecte les critères établis par Sterbenz, mais son temps d'exécution est long car il repose sur une comparaison de signes entre les deux entrées. Le second, relativement efficace, combine deux des implémentations proposées en section 4.1.1 et fournit l'arrondi correct de deux nombres flottants au format *binary64* :

```
1 double average(double C, double x, double y) {
2   if (C <= abs(x))
3     return x/2+y/2;
4   else
5     return (x+y)/2;
6 }
```

avec `C` une constante comprise entre 2^{-967} et 2^{970}.

Boldo prouve que ce programme fournit l'arrondi correct même en cas de dépassement de capacité inférieur et qu'aucun dépassement parasite de capacité supérieur ne se produit [24]. Ces résultats ne sont possibles que parce que les bornes sur les valeurs possibles de C sont bien choisies, la borne inférieure permettant d'éviter d'utiliser l'implémentation de la première branche lorsque des valeurs dénormalisées peuvent apparaître et la borne supérieure permettant de ne pas utiliser la seconde branche lorsqu'il y a un risque de dépassement parasite de capacité supérieur.

Goualard propose un algorithme comparable [88]. Néanmoins, il s'intéresse à un problème légèrement différent portant sur le calcul du point-milieu d'un intervalle, ce qui induit des différences dans le traitement des valeurs exceptionnelles. Il complète en outre le travail mené par Sterbenz en proposant quelques autres algorithmes dont il dresse une liste d'avantages et d'inconvénients :

- $(x \ominus (x \oslash 2)) \oplus (y \oslash 2)$, qui est moins précis en cas de dépassement de capacité inférieur, mais ne provoque pas de dépassement parasite de capacité supérieur.
- d'autres implémentations utilisant des modes d'arrondi dirigés, comme par exemple $\triangle(\triangle(x/2) + \triangle(y/2))$, avec $\triangle$ l'arrondi vers $+\infty$. La précédente formule ne respecte toutefois pas l'ensemble des critères de Sterbenz [195] : en effet, l'un de ces critères stipule que le point-milieu doit appartenir à l'intervalle, ce qui n'est pas forcément le cas ici.

Enfin, Kornerup *et al.* proposent un algorithme qui calcule l'arrondi correct de la moyenne de deux nombres x et y lorsque x et y sont du même ordre de grandeur [124]. Ce résultat reste d'ailleurs valable lorsque les nombres flottants considérés sont des nombres décimaux. Cependant, les hypothèses sur x et y sont assez restrictives : soit $0 \leq y \leq x \leq 2y$, soit $2y \leq x \leq y \leq 0$.

4.2 Algorithmes infructueux en base 10

D'un point de vue mathématique, *i.e.* lorsque les opérations et paramètres sont réels, les formules présentées en section 4.1.1 sont parmi les plus intuitives pour calculer la moyenne de deux nombres. En arithmétique flottante binaire, si nous faisons abstraction des comportements exceptionnels, certaines de ces implémentations fournissent

par ailleurs l'arrondi correct. Dans cette section, nous montrons qu'il est possible d'exhiber des contre-exemples assez simples lorsque les nombres flottants considérés sont décimaux. Pour des raisons de lisibilité dans la présentation des contre-exemples, nous utilisons un format décimal avec 4 chiffres de mantisse.

4.2.1 Formules basées sur $(x \oplus y) \oslash 2$

La formule la plus naïve est $(x \oplus y) \oslash 2$, qui fournit l'arrondi correct en base 2 en l'absence de dépassement de capacité supérieur. En base 10, le résultat de la division d'un nombre flottant par 2 est soit correct quand la mantisse est paire ou suffisamment petite, soit un point-milieu quand la mantisse est impaire (voir détails en section 4.3.1). Dans ce dernier cas, le choix de la règle de bris d'égalité du mode d'arrondi est alors déterminant. Ce constat rend aisé l'exhibition d'un contre-exemple : si l'un des nombres flottants est négligeable par rapport au second au moment de l'addition et que le résultat de la division par 2 n'est pas exact, le résultat obtenu est incorrect. Considérons par exemple $x = 3001 \times 10^{10}$ et $y = 1000 \times 10^{-10}$. L'arrondi de la somme de x et y est égal à x car y est petit par rapport à x et $x/2 = 1500{,}5 \times 10^{10}$. Avec un mode d'arrondi au plus proche avec règle de bris d'égalité vers le nombre pair (qui, ici, revient à arrondir vers $-\infty$), nous obtenons $(x \oplus y) \oslash 2 = 1500 \times 10^{10}$. Cependant, comme y est strictement positif, la valeur exacte de $(x + y)/2$ est légèrement supérieure au point-milieu $1500{,}5 \times 10^{10}$ et nous aurions dû arrondir vers $+\infty$ et obtenir le résultat correct 1501×10^{10}.

La formule $(x \oplus y) \otimes 5 \oslash 10$ est un bon candidat car elle est mathématiquement équivalente et fait apparaître une division par 10, exactement représentable en base 10. Cependant, le contre-exemple précédent invalide cette implémentation.

4.2.2 Formules basées sur $(x \oslash 2) \oplus (y \oslash 2)$

Afin d'éluder le problème de la sommation $x \oplus y$, nous étudions l'implémentation qui suit : $(x \oslash 2) \oplus (y \oslash 2)$. En base 2, elle fournit également l'arrondi correct en l'absence de dépassement de capacité inférieur. Malheureusement, pour des raisons similaires, cet algorithme ne fournit pas l'arrondi correct en base 10 pour le contre-exemple exhibé en section 4.2.1, *i.e.* $x = 3001 \times 10^{10}$ et $y = 1000 \times 10^{-10}$. En effet, $x/2$ étant un point-milieu, $(x \oslash 2) \oplus (y \oslash 2)$ n'est pas arrondi correctement.

L'utilisation d'un opérateur `FMA` (voir section 2.1.3) semble judicieuse puisqu'elle permettrait de se limiter à deux arrondis : $\circ(x \times 0.5 + (y \oslash 2))$. La division de y par 2 peut être inexacte, mais la division de x par 2 et la sommation finale ne nécessitent qu'un arrondi. Cependant, si cette idée permet d'obtenir l'arrondi correct pour l'exemple précédent, il est possible de trouver un contre-exemple invalidant cette nouvelle implémentation. Soit $x = 2001 \times 10^{10}$ et $y = 2001 \times 10^{8}$. Nous obtenons $y \oslash 2 = 1000 \times 10^{8}$, qui est le résultat de l'arrondi du point-milieu $y/2$. Cette valeur est ensuite ajoutée à $x/2 = 1000{,}5 \times 10^{10}$, ce qui donne le résultat $1010{,}5 \times 10^{10}$, dont l'arrondi est 1010×10^{10}. Cela diffère du résultat attendu, *i.e.* l'arrondi de $1010{,}505 \times 10^{10}$, *i.e.* 1011×10^{10}.

Bien qu'infructueuse sur cet exemple, cette dernière solution fournit l'arrondi correct pour une classe importante d'exemples. Nous l'utilisons donc, conjuguée à un test sur les valeurs dont on souhaite calculer la moyenne, dans les cas favorables (voir section 4.3).

4.3 Algorithme correct en arithmétique décimale

Nous proposons un algorithme en arrondi correct (algorithme 2) ne reposant que sur un test. Dans cette section, nous prouvons la correction de cet algorithme pour toute base paire et en particulier pour la base 10. La notation $\mathbb{F}$ désigne un format de nombres flottants en base paire, dont la précision p est strictement supérieure à 1 et tel que $e_{\min} < 0$ (ce qui dans le cas des bases paires implique que 0,5 est représentable dans $\mathbb{F}$). Le mode d'arrondi considéré est l'arrondi au plus proche noté $\circ$ avec règle de bris d'égalité quelconque. Nous supposons également que l'opération `FMA` est disponible.

```
1 Function TwoSum(x, y)
2     s ← ∘(x + y)
3     x' ← ∘(s − y)
4     y' ← ∘(s − x')
5     δa ← ∘(x − x')
6     δb ← ∘(y − y')
7     r ← ∘(δa + δb)
8     return (r, s)
```

Algorithme 1 : Algorithme TwoSum

```
1 Function Average10(x, y)
2     (a, b) = TwoSum (x, y)
3     if ∘τ(a × 0,5 − (a ⊘ 2)) = 0 then
4         return ∘τ(b × 0,5 + (a ⊘ 2))
5     else
6         return ∘τ(a × 0,5 + b)
```

Algorithme 2 : Algorithme de calcul de moyenne de deux nombres décimaux

L'algorithme utilise l'opérateur TwoSum [58, 123], une transformation exacte calculant la somme arrondie a de deux nombres flottants x et y et l'erreur induite par cette sommation (qui est exactement représentable dans le format $\mathbb{F}$ et est notée b dans le reste de la section). L'opérateur TwoSum est décrit par l'algorithme 1. Par définition, l'opérateur TwoSum vérifie $(a, b) = \text{TwoSum}(x, y) \implies x + y = a + b$. Calculer la moyenne de x et y revient donc à calculer la moyenne de a et b. Comme b est l'erreur d'arrondi causée par la sommation, nous obtenons une information supplémentaire, à

savoir que $|b| \leq \frac{\text{ulp}(a)}{2}$. Cette hypothèse permet de prouver la propriété d'arrondi correct quelle que soit la branche empruntée après le test de la ligne 3 (voir détails dans les sections 4.3.1 et 4.3.3).

L'algorithme 2 ne fait appel qu'à un unique test et comporte deux branches. La première branche est empruntée lorsque la division de a par 2 est représentable dans le format $\mathbb{F}$, ce qui est toujours vrai lorsque la mantisse de a est paire mais est également vérifié pour certains nombres de mantisse impaire. La seconde branche est quant à elle empruntée lorsque a est un nombre impaire dont la division par 2 n'appartient pas à $\mathbb{F}$.

4.3.1 Arrondi correct dans le cas où a est strictement positif

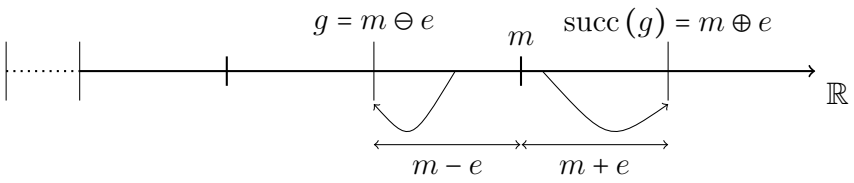
Dans cette section, nous supposons que a est strictement positif, la généralisation au cas négatif est naturelle par argument de symétrie et est présentée en section 4.3.3 (premier paragraphe). Par ailleurs, nous supposons les exposants non bornés, ce qui correspond au format `FLX` de Flocq (voir section 2.2.4). La prise en compte des dépassements graduels de capacité inférieurs sera présentée dans le second paragraphe de la section 4.3.3. La preuve repose sur plusieurs résultats techniques, que nous commençons par démontrer (voir section 2.1.2 pour la définition de $\text{succ}(g)$ utilisé ci-dessous).

Lemme 4.1. *Soit* $m = g + \frac{1}{2} \times \text{ulp}(g)$ *avec* $g \in \mathbb{F}$, $g > 0$ *et* $0 < e \leq \frac{\text{ulp}(g)}{2}$.

- $m \ominus e = g$
- $m \oplus e = \text{succ}(g)$

Démonstration. La preuve repose sur le fait que m soit le point-milieu entre g et $\text{succ}(g)$ et sur le fait que e soit positif et suffisamment petit. Cela peut s'expliquer de façon intuitive en observant les valeurs respectives des différentes variables, comme indiqué sur la figure 4.1. □

FIGURE 4.1 – Disposition des valeurs utilisées par le lemme 4.1



Le lemme 4.2 établit les propriétés d'un nombre flottant qui ne peut pas être divisé par 2 de façon exacte :

Lemme 4.2. *Soit* $x \in \mathbb{F}$.

$$x/2 \notin \mathbb{F} \implies |M_x| \geq 2 \times 10^{p-1} + 1 \wedge M_x \in (2\mathbb{Z} + 1).$$

Démonstration. Soit $x \in \mathbb{F}$ tel que $x/2 \notin \mathbb{F}$.

Par contraposition de la seconde moitié de la conclusion, nous supposons que $M_x \in 2\mathbb{Z}$, donc $\frac{M_x}{2} \in \mathbb{Z}$.
Soit $n = \frac{M_x}{2}$, $\frac{x}{2} = \frac{M_x}{2} \times 10^{E_x} = n \times 10^{E_x}$. Nous avons $x \in \mathbb{F}$, donc $|M_x| < 10^p$, ce qui implique que $|n| < 10^p$. Nous en déduisons que $\frac{x}{2} \in \mathbb{F}$ car $\frac{x}{2} = n \times 10^{E_x}$ avec :

$$n \in \mathbb{Z} \wedge E_x \in \mathbb{Z} \wedge |n| < 10^p.$$

Ainsi, la première hypothèse $M_x \in 2\mathbb{Z}$ est fausse car elle contredit le fait que $\frac{x}{2} \notin \mathbb{F}$, donc nous avons $M_x \in (2\mathbb{Z}+1)$.

Supposons maintenant que $|M_x| < 2 \times 10^{p-1}$.
Nous avons $M_x \in (2\mathbb{Z}+1)$, donc $M_x - 1 \in 2\mathbb{Z}$ et $\frac{M_x-1}{2} \in \mathbb{Z}$. Soit $n = 10 \times \frac{M_x}{2}$, donc $\frac{x}{2} = n \times 10^{E_x-1}$. À partir de l'égalité $\frac{M_x}{2} = \frac{M_x-1}{2} + \frac{1}{2}$, nous prouvons que $n \in \mathbb{Z}$:

$$n = 10 \times \left(\frac{M_x - 1}{2} + \frac{1}{2}\right) = 10 \times \frac{M_x - 1}{2} + 5.$$

De plus, par l'hypothèse $|M_x| < 2 \times 10^{p-1}$, nous bornons la valeur de n : $\left|\frac{M_x}{2}\right| < 10^{p-1}$, donc $|n| < 10^p$. Donc $\frac{x}{2} \in \mathbb{F}$ car $\frac{x}{2} = n \times 10^{E_x-1}$ avec :

$$n \in \mathbb{Z} \wedge E_x - 1 \in \mathbb{Z} \wedge |n| < 10^p.$$

Une fois encore, par contradiction, l'hypothèse $|M_x| < 2 \times 10^{p-1}$ est fausse, donc $|M_x| \geq 2 \times 10^{p-1}$.

Nous avons $M_x \in (2\mathbb{Z}+1) \wedge |M_x| \geq 2 \times 10^{p-1}$, mais $2 \times 10^{p-1} \in 2\mathbb{Z}$, donc $|M_x| > 2 \times 10^{p-1}$ et ainsi $M_x \in (2\mathbb{Z}+1) \wedge |M_x| \geq 2 \times 10^{p-1} + 1$. □

Le théorème[2] qui suit établit la propriété d'arrondi correct de l'algorithme 2.

Théorème 4.3. *Supposons que $p \geqslant 2$. Soient $a, b \in \mathbb{F}$ tels que $|b| \leq \frac{1}{2} \times \mathrm{ulp}(a)$ et $a > 0$:*

- *Si $\frac{a}{2} \in \mathbb{F}$:*

$$\circ^\tau\left(\frac{a+b}{2}\right) = \circ^\tau(b \times 0.5 + a \oslash 2).$$

- *Si $\frac{a}{2} \notin \mathbb{F}$:*

$$\circ^\tau\left(\frac{a+b}{2}\right) = \circ^\tau(a \times 0.5 + b).$$

2. En fait, ce théorème ne traduit cette propriété que si le test de la ligne 3 de l'algorithme 2 est équivalent à $\frac{a}{2} \in \mathbb{F}$: ce point est discuté dans la suite de la présente section.

Démonstration. Soit $a \in \mathbb{F}$ et $b \in \mathbb{F}$ tels que $|b| \leq \frac{1}{2} \times \text{ulp}(a)$ et $a > 0$.

- Si $b = 0$:

$$\circ^\tau\left(\frac{a}{2} + b\right) = \circ^\tau\left(\frac{a}{2} + \frac{b}{2}\right) = \circ^\tau\left(\frac{a}{2}\right).$$

Donc, nous pouvons nous restreindre au cas où $b \neq 0$ dans le reste de la démonstration, pour les deux cas qui suivent :

- Premier cas : $\frac{a}{2} \in \mathbb{F}$,

$$\circ^\tau\left(b \times 0,5 + a \oslash 2\right) = \circ^\tau\left(\frac{b}{2} + \frac{a}{2}\right) = \circ^\tau\left(\frac{a+b}{2}\right).$$

- Second cas : $\frac{a}{2} \notin \mathbb{F}$,
D'après l'égalité $\frac{M_a}{2} = \frac{M_a - 1}{2} + \frac{1}{2}$, nous avons $\frac{a}{2} = \left(\frac{M_a - 1}{2} + \frac{1}{2}\right) \times 10^{E_a}$.
Par application du lemme 4.2, nous avons $M_a \in (2\mathbb{Z} + 1)$, donc $M_a - 1 \in 2\mathbb{Z}$ et $\frac{M_a - 1}{2} \in \mathbb{Z}$.
Ce lemme établit également que $|M_a| \geq 2 \times 10^{p-1} + 1$, donc :

$$\begin{aligned}
& 2 \times 10^{p-1} + 1 \leq |M_a| < 10^p \\
\Longrightarrow\ & 2 \times 10^{p-1} \leq |M_a - 1| < 10^p + 1 \\
\Longrightarrow\ & 10^{p-1} \leq \left|\frac{M_a - 1}{2}\right| < \frac{10^p + 1}{2} \\
\Longrightarrow\ & 10^{p-1} \leq \left|\frac{M_a - 1}{2}\right| < 10^p.
\end{aligned}$$

Soit $c = \frac{M_a - 1}{2} \times 10^{E_a}$, nous avons $c \in \mathbb{F}$ car :

$$\frac{M_a - 1}{2} \in \mathbb{Z} \wedge E_a \in \mathbb{Z} \wedge \left|\frac{M_a - 1}{2}\right| < 10^p.$$

Nous avons aussi $\text{ulp}(a) = \text{ulp}(c)$ car $E_a = E_c$, $10^{p-1} \leq M_a < 10^p$ et $10^{p-1} \leq M_c < 10^p$. Nous pouvons réécrire $\frac{a}{2}$ comme :

$$\frac{a}{2} = c + \frac{1}{2} \times 10^{E_a} = c + \frac{1}{2} \times \text{ulp}(c).$$

- Si $b > 0$, nous avons : $0 < b \leq \frac{1}{2} \times \text{ulp}(a)$. Donc $0 < \frac{b}{2} \leq \frac{1}{2} \times \text{ulp}(a)$. Ainsi, d'après le lemme 4.1 :

$$\circ^\tau\left(\frac{a}{2} + b\right) = \circ^\tau\left(\frac{a}{2} + \frac{b}{2}\right) = \text{succ}(c).$$

- Si $b < 0$, nous avons : $0 < -b \leq \frac{1}{2} \times \text{ulp}(a)$. Donc $0 < -\frac{b}{2} \leq \frac{1}{2} \times \text{ulp}(a)$. Ainsi, par application du lemme 4.1 :

$$\circ^\tau\left(\frac{a}{2} - (-b)\right) = \circ^\tau\left(\frac{a}{2} - \left(-\frac{b}{2}\right)\right) = c.$$

Donc, dans tous les cas, nous avons :

$$\circ^{\tau}\left(\frac{a}{2}+b\right)=\circ^{\tau}\left(\frac{a}{2}+\frac{b}{2}\right).$$

□

Pour que le théorème 4.3 soit une preuve de correction de l'algorithme 2, il reste à prouver que $\circ^{\tau}(a\times 0.5-(a\oslash 2))=0$ correspond bien à tester si $a/2\in\mathbb{F}$. Dans ce cas, comme les exposants sont non bornés, ces deux tests sont équivalents à $a/2=a\oslash 2$.

Il faut noter, à la ligne 6 de l'algorithme 2, que b n'est pas divisé par 2. En pratique, seul le signe de b nous intéresse. Par conséquent, nous aurions pu utiliser aussi bien b que $b\oslash 2$. Nous choisissons d'utiliser b pour deux raisons. Premièrement, cela permet de se passer d'une opération flottante. Deuxièmement, lorsque le dépassement graduel de capacité inférieur sera pris en compte, il deviendra nécessaire de ne pas diviser b par 2. En effet, lorsque b est très petit, nous pourrions avoir $b\oslash 2=0$ et ainsi perdre l'information sur le signe de b (voir détails dans la section 4.3.3).

4.3.2 Formalisation des résultats

Les démonstrations des résultats présentés en section 4.3.1 sont particulièrement difficiles à vérifier et hautement perméables, sinon aux raisonnements faux, au moins à l'oubli de cas ou à l'utilisation d'hypothèses erronées. C'est pourquoi nous formalisons l'ensemble du raisonnement dans l'assistant de preuves Coq.

L'opérateur TwoSum [58, 123] est utilisé à la ligne 2 de l'algorithme 2 afin de transformer les entrées. Bien qu'il existe une formalisation de TwoSum dans Coq, nous nous limitons à prouver la correction de l'algorithme appliqué aux sorties de TwoSum. Nous supposons que la valeur absolue de la seconde entrée est inférieure à la moitié de l'*ulp* de la première entrée, *i.e.* `Rabs b <= ulp a / 2`. Comme expliqué en section 4.3.1, cette hypothèse est cruciale pour prouver que l'algorithme fournit l'arrondi correct de la moyenne des deux entrées mais n'implique pas $a=\circ(a+b)$.

En Coq, l'algorithme 2 de la page 61 est défini comme suit :

```
Variable choice : Z → bool.
Notation round_flx := (round radix10 (FLX_exp prec) (Znearest choice)).
( ... )
Definition average10 (a b : R) :=
  if (Req_bool (round_flx (a/2 - round_flx (a/2))) 0)
    then round_flx (b/2 + round_flx (a/2))
    else round_flx (a/2 + b).
```

Dans le code Coq ci-dessus :

- `Req_bool u v` est à valeur dans `bool` et renvoie vrai si $u=v$, faux sinon ;
- `round_flx` désigne l'arrondi au plus proche (`Znearest`), avec règle de bris d'égalité quelconque (`choice`) [3], dans le format `FLX` de Flocq (voir section 2.2.4).

3. En d'autres termes `Znearest choice` désigne $\circ^{\tau}$.

La propriété d'arrondi correct que nous cherchons à démontrer peut être spécifiée de la façon suivante :

```
Notation format := (generic_format radix10 (FLX_exp prec)).
( ... )
Theorem average10_correct : forall a b, format a → format b →
           Rabs b <= ulp a / 2 → average10 a b = round ((a+b)/2).
```

Dans le code Coq ci-dessus, `format` désigne le prédicat vérifiant si un nombre est représentable dans le format flottant décimal `FLX` de Flocq (voir section 2.2.4).

La démonstration formelle du théorème Coq `average10_correct` suit globalement la démonstration papier du théorème 4.3 et n'est pas extrêmement difficile. Il a néanmoins d'abord fallu formaliser la preuve des résultats intermédiaires, *i.e.* les lemmes 4.1 et 4.2, dont la démonstration s'est révélée plus ardue du fait de certains aspects techniques propres à la bibliothèque Flocq, *e.g.* le concept d'exposant canonique [38].

La preuve formelle présentée dans la section courante se restreint au cas où a est strictement positif et où le format utilisé est le format `FLX` (voir section 2.2.4). La section 4.3.3 présente les généralisations que nous avons menées avec l'aide de Coq.

4.3.3 Généralisations

Cas où a (premier opérande de la moyenne) est négatif

Nous avons prouvé que l'algorithme 2 implémenté suivant un mode d'arrondi $\circ^\tau$, *i.e.* au plus proche avec règle de bris d'égalité τ quelconque, fournissait l'arrondi correct. Notons average10$^\tau$ l'expression mathématique de cet algorithme.

Nous prouvons l'existence d'une fonction $\circ^{\tilde{\tau}} : \mathbb{R} \to \mathbb{F}$ (paramétrée par une fonction $\tilde{\tau} : \mathbb{Z} \to \text{bool}$) telle que pour tout $x \in \mathbb{R}, \circ^{\tilde{\tau}}(x) = -\circ^\tau(-x)$ et qui implémente bien un arrondi au plus proche avec règle de bris d'égalité $\tilde{\tau}$.

Supposons que $a < 0$ (a étant le premier opérande de la moyenne). Comme le théorème 4.3 est valide pour toute règle de bris d'égalité et que $0 \leqslant -a$, nous avons average10$^{\tilde{\tau}}(-a,-b) = \circ^{\tilde{\tau}}((-a-b)/2)$. Donc, par définition de $\circ^{\tilde{\tau}}$, average10$^{\tilde{\tau}}(-a,-b) = -\circ^\tau((a+b)/2)$. Nous utilisons ensuite Coq pour démontrer que average10$^{\tilde{\tau}}(-a,-b) =$ $-$average10$^\tau(a,b)$ et obtenons ainsi average10$^\tau(a,b) = \circ^\tau((a+b)/2)$.

Si $a = 0$, alors $b = 0$ et average10$^\tau(a,b) = 0$, qui est bien l'arrondi correct attendu.

Dépassement graduel de capacité inférieur

Nous prenons ensuite en compte les dépassements graduels de capacité inférieurs. Le passage du format `FLX` au format `FLT` conduit à distinguer deux cas :

- si $b = 0$, alors a peut potentiellement être dénormalisé. Dans le format `FLX`, $a \neq 0 \Rightarrow o(a) \neq 0$, ce qui n'est pas nécessairement vrai dans le format `FLT`, *e.g.* lorsque a est inférieur à la moitié du successeur de 0. Ainsi, le test réalisé à la ligne 3 de l'algorithme 2 ne correspond pas à tester si $a/2$ est représentable dans

le format `FLT` considéré. Néanmoins, comme $b = 0$, cela n'a guère d'importance, l'arrondi correct étant trivialement obtenu quelle que soit la branche empruntée.
- si $b \neq 0$, par l'hypothèse $|b| \leq \frac{\mathrm{ulp}(a)}{2}$, a est nécessairement supérieur à $10^{p+e_{\min}}$ ce qui implique que a, $a/2$, et $a \oslash 2$ sont normalisés. Par conséquent, le test de la ligne 3 de l'algorithme 2 a le même sens que dans le format `FLX` et vérifie si $a/2$ est représentable. Ainsi, les démonstrations fournies en section 4.3.1 restent valables. Par ailleurs, lorsque b est dénormalisé, l'information sur son signe pourrait être perdue lors du calcul de $b/2$ (pour $b = \mathrm{succ}(0)$). Cependant, b n'est pas divisé par 2 à la ligne 6 de l'algorithme 2, ce qui permet de contourner ce problème.

Généralisation à toute base paire

Nous n'utilisons aucune propriété propre à la base 10, sinon sa parité. Le passage de la base 10 à une base paire quelconque s'est fait sans encombre et n'a généré aucun but nouveau (à l'exception de quelques rudiments d'arithmétique entière que Coq démontre automatiquement lorsque la base est fixée et nécessitant une étape de réécriture supplémentaire lorsqu'il s'agit d'un paramètre, suivie de l'utilisation d'une tactique automatique). L'algorithme 2 fournit donc l'arrondi correct pour toute base paire.

4.4 Conclusion

Aussi simple soit l'idée de calculer la moyenne de deux nombres flottants, elle a permis de montrer qu'une propriété valide dans un format de représentation binaire n'est pas aisément généralisable à toute base. Nous avons montré que les algorithmes fournissant l'arrondi correct de la moyenne en base 2 échouaient en base 10. Il a fallu construire un programme plus élaboré et relativement plus coûteux puisqu'il repose sur un branchement conditionnel et sur l'utilisation judicieuse de l'opérateur TwoSum. Cependant, nous avons pu prouver qu'il calculait l'arrondi correct en tenant compte des dépassements inférieurs et pour une base paire quelconque.

Bien que lisible et relativement courte, la preuve de correction proposée est technique. L'utilisation de l'assistant de preuves Coq a permis d'une part de garantir la validité de la démonstration et d'autre part de guider la généralisation des résultats. La preuve formelle est concise (700 lignes de code Coq) et de taille comparable à la démonstration formelle de l'algorithme proposé en base 2 par Boldo [24].

En revanche, nous ne gérons pas les dépassements de capacité supérieurs. S'il est possible de montrer que l'algorithme proposé renvoie une valeur finie pourvu que TwoSum renvoie une valeur finie, ce dernier peut dans de rares cas être responsable de dépassements parasites de capacité supérieurs [33].

Par ailleurs, il ne serait pas simple de généraliser l'algorithme au calcul de moyenne dans le cas général, *i.e.* $\frac{X_1+X_2+\cdots+X_n}{n}$ avec $X_1, X_2, \ldots, X_n$ des nombres flottants décimaux. Il n'est en effet pas toujours possible de calculer l'arrondi correct d'une somme de 3

nombres flottants ou plus [4].

L'une des faiblesses de l'algorithme proposé est l'utilisation de l'algorithme TwoSum, qui comporte six opérations flottantes. Il semble toutefois difficile de mettre en place un algorithme correct sans moyen de comparer les ordres de grandeurs des deux opérandes, sinon au prix de nombreux branchements conditionnels, qui seraient plus coûteux que l'algorithme TwoSum lui-même.

Nous pourrions également envisager d'étendre l'approche aux bases impaires. Cette perspective présente cependant des difficultés techniques. En effet, le raisonnement actuel ne peut être réutilisé car nous utilisons la parité de la base à plusieurs reprises. La gestion des bases impaires est par ailleurs difficile à cause de la gestion des points-milieux et des règles de bris d'égalité, qui n'est pas aussi standard qu'en base paire.

Cet exemple met en exergue la nécessité de démonstrations rigoureuses pour les implémentations en précision finie, même les plus simples [5]. Lorsque ces démonstrations sont techniques ou fastidieuses, nous pensons que l'utilisation d'assistants à la preuve formelle tels que Coq apporte une garantie supplémentaire quant à la correction des raisonnements. Ce parti-pris sera le socle commun à l'ensemble des travaux présentés dans ce manuscrit.

4. Il serait éventuellement possible d'utiliser un accumulateur long de Kulisch [126].

5. Sterbenz évoque la nécessité d'implémenter des programmes minutieusement écrits (« *carefully written programs* ») par opposition aux programmes écrits « vite et mal » (« *quick and dirty programs* ») [195].

Deuxième partie

Erreurs d'arrondi de méthodes de Runge-Kutta

Chapitre 5

Erreurs d'arrondi des méthodes de Runge-Kutta : cas linéaire scalaire

La résolution exacte des équations différentielles ordinaires n'étant pas toujours possible, de nombreuses méthodes de résolution numériques ont été mises au point [59] (voir chapitre 2 pour une présentation de ces méthodes). Les méthodes de Runge-Kutta explicites [45] figurent parmi les schémas numériques les plus couramment utilisés, en particulier la méthode d'Euler et la méthode de Runge-Kutta d'ordre 4. À titre d'exemple, parmi les 13 méthodes d'intégration numérique qu'embarque l'outil Simulink [1], 11 méthodes appartiennent à la classe des schémas de Runge-Kutta.

Les méthodes d'intégration numérique ont été construites dans l'objectif d'obtenir une bonne approximation de la solution exacte et sont inextricablement liées aux erreurs de méthodes qu'elles induisent, bien connues et intensivement étudiées. L'un des aspects les plus importants de la recherche en analyse numérique des équations différentielles a d'ailleurs pour objectif d'augmenter l'ordre des méthodes sans compromis sur l'efficacité [59, 45, 98]. L'implémentation de méthodes numériques itératives en arithmétique à virgule flottante induit une seconde source d'erreur, à savoir les erreurs d'arrondi pouvant s'accumuler au cours des itérations et fausser les résultats obtenus. Cependant, les erreurs d'arrondi étant (observationnellement) négligeables par rapport aux erreurs de méthode [175], elles ont rarement fait le sujet d'études poussées. Des travaux se sont néanmoins portés sur ces problématiques, adoptant des approches très diverses. Le chapitre 3 de ce manuscrit propose un état de l'art de ces différents travaux.

Dans le cas des systèmes linéaires, les solutions peuvent être exprimées de manière exacte. Cependant, l'utilisation de méthodes numériques itératives est parfois privilégiée, notamment lorsque l'objectif est d'observer l'évolution temporelle de la solution. La résolution explicite des systèmes linéaires utilise en effet l'exponentielle de matrice, dont le calcul efficace et précis s'avère être un problème difficile [84, 151], reposant généralement sur des approches telles que les approximants de Padé [10]. Certains outils comme Simulink embarquent par ailleurs des solveurs à base de méthodes de Runge-

1. Simulink est un outil permettant de modéliser et simuler des systèmes dynamiques : `https://fr.mathworks.com/products/simulink.html`

Kutta et ne traitent pas à part les systèmes linéaires.

Dans cette seconde partie du manuscrit, nous nous intéressons aux erreurs d'arrondi induites par l'implémentation de méthodes de Runge-Kutta explicites appliquées aux systèmes linéaires. Nous proposons une méthodologie mécanique, à grains fins, et qui tire parti de certaines propriétés mathématiques des méthodes numériques, comme la stabilité linéaire [129]. Nous tenons compte des éventuels comportements exceptionnels d'arithmétique à virgule flottante, à savoir les dépassements de capacité inférieurs et supérieurs. L'analyse générique que nous présentons sera instanciée à trois méthodes classiques : la méthode d'Euler, la méthode de Runge-Kutta d'ordre 2 et la méthode de Runge-Kutta d'ordre 4.

Ce chapitre, qui permet d'introduire notre approche ainsi qu'un ensemble de définitions, est plus particulièrement voué au cas des systèmes les plus simples, à savoir les systèmes linéaires scalaires (unidimensionnels). En section 5.1, nous présentons les systèmes, les méthodes et les implémentations étudiés. La méthodologie utilisée pour borner les erreurs d'arrondi (basée sur la distinction entre les erreurs dites locales et globales) est présentée en section 5.2. Nous montrons ensuite que, dans le cas des méthodes stables, les solutions calculées sont dans la plupart des cas décroissantes (section 5.3). En section 5.4, nous bornons les erreurs d'arrondi locales, *i.e.* commises au cours d'un pas d'itération de la méthode. Puis, en section 5.5, à partir des bornes sur les erreurs locales, nous bornons l'erreur globale, *i.e.* résultant de l'accumulation des erreurs d'arrondi. Enfin, en section 5.6, nous prouvons qu'une condition « réaliste » sur la valeur de la solution initiale suffit à se prémunir des dépassements de capacité supérieurs.

Remarque 1. *Les travaux présentés dans ce chapitre ont fait l'objet d'un premier article ne couvrant pas la gestion des dépassements de capacité [29] et d'un second article qui raffine les bornes et présente la gestion des comportements exceptionnels [30].*

Remarque 2. *Les résultats présentés dans ce chapitre n'ont pas directement fait l'objet d'une formalisation car ils peuvent pour la plupart être obtenus comme instances des résultats du chapitre 6 (cas linéaire matriciel), qui sont pour leur part formalisés.*

5.1 Systèmes et méthodes étudiés

5.1.1 Résolution numérique des systèmes linéaires scalaires par des méthodes de Runge-Kutta

La famille la plus simple de problèmes avec condition initiale est associée aux équations différentielles ordinaires de la forme :

$$\dot{y} = \lambda y \tag{5.1}$$

avec $\lambda \in \mathbb{C}$, dont la solution explicite est $t \mapsto y_0 e^{\lambda(t-t_0)}$ lorsque $y(t_0) = y_0$.

Une telle équation décrit un problème stable, *i.e.* dont la solution converge vers zéro, si et seulement si $\Re(\lambda) < 0$, avec $\Re$ la partie réelle de λ. Dans la suite de ce chapitre,

nous nous intéressons au cas particulier où $\lambda \in \mathbb{R}$, dont la condition de stabilité se réduit à $\lambda < 0$. L'application d'une méthode de Runge-Kutta explicite sur le problème linéaire défini par l'équation (5.1) définit la relation :

$$y_{n+1} = R(h, \lambda) y_n. \tag{5.2}$$

Parmi les méthodes de Runge-Kutta explicites les plus fréquemment utilisées (voir section 2.3.3), la méthode d'Euler (voir figure 2.3) est l'exemple le plus simple :

Exemple 5.1. *Application de la méthode d'Euler explicite sur l'équation* (5.1)

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda) y_n = R_{Euler}(h, \lambda) y_n.$$

Nous nous intéressons également à l'application des méthodes de Runge-Kutta d'ordre 2 (voir figure 2.4) et d'ordre 4 (voir figure 2.2(a)) sur l'équation (5.1) :

Exemple 5.2. *Application de la méthode RK2 explicite sur l'équation* (5.1)

$$k_1 = \lambda y_n \qquad k_2 = \lambda(y_n + \frac{1}{2} h k_1) = (\lambda + \frac{1}{2} h \lambda^2) y_n$$

$$y_{n+1} = y_n + h k_2 = (1 + h\lambda + \frac{1}{2} h^2 \lambda^2) y_n = R_{RK2}(h, \lambda) y_n. \tag{5.3}$$

Exemple 5.3. *Application de la méthode RK4 explicite sur l'équation* (5.1)

$$\begin{aligned} k_1 &= \lambda y_n & k_2 &= \lambda(y_n + \frac{1}{2} h k_1) = (\lambda + \frac{1}{2} h \lambda^2) y_n \\ k_3 &= \lambda(y_n + \frac{1}{2} h k_2) = (\lambda + \frac{1}{2} h \lambda^2 + \frac{1}{4} h^2 \lambda^3) y_n \\ k_4 &= \lambda(y_n + h k_3) = (\lambda + h\lambda^2 + \frac{1}{2} h^2 \lambda^3 + \frac{1}{4} h^3 \lambda^4) y_n \end{aligned}$$

$$\begin{aligned} y_{n+1} &= y_n + h(\frac{1}{6} k_1 + \frac{1}{3} k_2 + \frac{1}{3} k_3 + \frac{1}{6} k_4) \\ &= \left(1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3 + \frac{1}{24}(h\lambda)^4\right) y_n = R_{RK4}(h, \lambda) y_n. \end{aligned} \tag{5.4}$$

Il convient, pour résoudre un problème stable, d'utiliser une méthode dite stable pour ce problème (voir chapitre 2). L'équation (5.2), enrichie d'une condition initiale, définit une suite géométrique, convergente pour $|R(h, \lambda)| < 1$. La fonction R est appelé *fonction de stabilité linéaire* de la méthode de Runge-Kutta [129]. Dans le cas des méthodes de Runge-Kutta explicites, il s'agit d'un polynôme en $h\lambda$. La région de stabilité associée à l'équation (5.2) (et donc associée à un couple système-méthode) détermine les valeurs que peut prendre $h\lambda$ pour satisfaire la propriété de stabilité de la méthode, *i.e.* tel que $|R(h, \lambda)| < 1$. La figure 5.1 représente les régions de stabilité pour différentes méthodes de Runge-Kutta explicites dans le cas où λ est complexe [2].

Dans la suite de ce chapitre, nous nous intéresserons plus particulièrement aux méthodes stables (au sens de la stabilité linéaire).

2. Dans le cas réel, il suffit de regarder l'axe des abscisses.

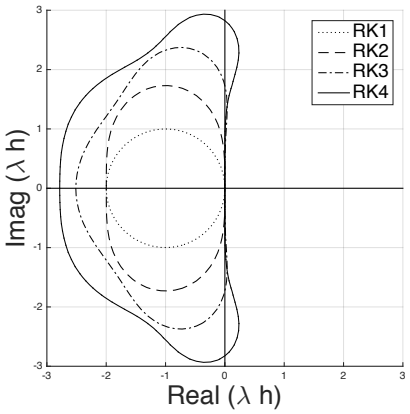


FIGURE 5.1 – Région de stabilité des méthodes de Runge-Kutta explicites de l'ordre 1 (*i.e.* Euler) à l'ordre 4. L'image est inspirée de l'ouvrage de Lambert [129].

5.1.2 Implémentations des méthodes de Runge-Kutta

La fonction R de l'équation (5.2) dépend uniquement de la méthode numérique employée et est purement « mathématique ». Il convient de distinguer la fonction R et l'implémentation correspondante en arithmétique à virgule flottante, notée $\widetilde{R}$.

$\widetilde{R}$ dépend de trois paramètres. Le premier d'entre eux est le pas h. Son choix revient à l'utilisateur et nous le considérons comme représentable dans le format flottant [3]. Le second paramètre de l'implémentation $\widetilde{R}$ est l'arrondi $\widetilde{\lambda}$ du coefficient λ, qui, en tant que donnée du système dynamique, pourrait ne pas être représentable. Contrairement à R, $\widetilde{R}$ dépend de l'itération courante n et ne peut pas être défini indépendamment de la solution calculée *via* l'implémentation $\widetilde{R}$ à l'itération précédente [4], notée $\widetilde{y_{n-1}}$. Une implémentation de la méthode de Runge-Kutta (correspondant à l'équation (5.2)) est donc de la forme :

$$\widetilde{y_0} \approx y_0 \in \mathbb{R} \qquad \forall n, \widetilde{y_{n+1}} = \widetilde{R}(h, \widetilde{\lambda}, \widetilde{y_n}). \tag{5.5}$$

Chaque méthode numérique peut être associée à une multitude d'implémentations qui ne sont pas équivalentes lorsque la précision de calcul est finie. Une implémentation naturelle consiste à considérer le polynôme $R(h, \lambda)$ puis à remplacer les opérations réelles par des opérations flottantes (*modulo* le choix de l'ordre d'évaluation). Par exemple, pour $RK4$ (voir $R_{RK4}(h, \lambda)$, équation (5.4)), l'implémentation « mathématique » est $\widetilde{R_{RK4}}(h, \widetilde{\lambda}, \widetilde{y_n}) = \circ\left[\widetilde{y_n} + h\widetilde{\lambda}\widetilde{y_n} + hh\frac{1}{2}\widetilde{\lambda}\widetilde{\lambda}\widetilde{y_n} + hhh\frac{1}{6}\widetilde{\lambda}\widetilde{\lambda}\widetilde{\lambda}\widetilde{y_n} + hhhh\frac{1}{24}\widetilde{\lambda}\widetilde{\lambda}\widetilde{\lambda}\widetilde{\lambda}\widetilde{y_n}\right]$ (pour rappel, $\circ[\dots]$ signifie que toutes les opérations à l'intérieur des crochets sont arrondies, avec un parenthésage implicite de la droite vers la gauche (voir section 2.1.3)).

3. En effet, seul l'ordre de grandeur de h compte. En d'autres termes, l'utilisateur choisit un pas h suffisamment petit pour répondre aux critères de précision souhaités pour l'approximation de la solution exacte. Si l'utilisateur choisit un pas d'intégration non représentable, le nombre flottant le plus proche est généralement tout aussi acceptable pour répondre aux critères imposés : nous travaillons donc directement avec ce nombre flottant.

4. Cela s'explique par le fait que les opérations flottantes ne sont pas nécessairement associatives.

Proposer une analyse d'erreurs d'arrondi pour cette implémentation que nous qualifierons d'implémentation « en forme mathématique » nous paraît cependant réducteur. En pratique, il arrive que l'implémentation d'une méthode de Runge-Kutta soit générée automatiquement sur la donnée du système à résoudre et du tableau de Butcher de la méthode (*e.g.* dans l'outil Simulink). Les implémentations obtenues font généralement apparaître un nombre d'opérations flottantes pouvant être important et peuvent donc s'avérer sensibles aux erreurs d'arrondi.

Nous nous intéressons à une classe d'implémentations que nous qualifierons d'implémentations « naïves ». Elles consistent à d'abord construire un à un les termes k_i résultant du tableau de Butcher de la méthode numérique (voir les exemples 5.2 et 5.3 pour RK2 et RK4) puis à sommer l'ensemble des k_i. Dans le cas de la méthode d'Euler, l'implémentation naïve est proche d'une implémentation en forme normale, *i.e.* $\widetilde{R_{Euler}}(h, \widetilde{\lambda}, \widetilde{y_n}) = \widetilde{y_n} \oplus h \otimes \widetilde{\lambda} \otimes \widetilde{y_n}$. Dans le cas des méthodes d'ordre supérieur, les implémentations naïves utilisent un nombre conséquent d'opérations flottantes :

Exemple 5.4. *L'application de la méthode RK2 sur l'équation différentielle avec condition initiale définie par l'équation* (5.1) *conduit à l'implémentation naïve :*

$$\widetilde{y_{n+1}} = \widetilde{R_{\mathrm{RK2}}}(h, \widetilde{\lambda}, \widetilde{y_n}) = \circ\left[\widetilde{y_n} + h\widetilde{\lambda}\widetilde{y_n} + hh\frac{1}{2}\widetilde{\lambda}\widetilde{\lambda}\widetilde{y_n}\right].$$

Exemple 5.5. *L'application de la méthode RK4 sur l'équation différentielle avec condition initiale définie par l'équation* (5.1) *conduit à l'implémentation naïve :*

$$\begin{aligned}\widetilde{y_{n+1}} = \widetilde{R_{\mathrm{RK4}}}(h, \widetilde{\lambda}, \widetilde{y_n}) = \circ\Bigg[&\widetilde{y_n} + \frac{h}{6}\widetilde{\lambda}\widetilde{y_n} + \frac{h}{3}\widetilde{\lambda}\widetilde{y_n} + \frac{h^2}{6}\widetilde{\lambda}^2\widetilde{y_n} + \frac{h}{3}\widetilde{\lambda}\widetilde{y_n} \\ &+\frac{h^2}{6}\widetilde{\lambda}^2\widetilde{y_n} + \frac{h^3}{12}\widetilde{\lambda}^3\widetilde{y_n} + \frac{h}{6}\widetilde{\lambda}\widetilde{y_n} + \frac{h^2}{6}\widetilde{\lambda}^2\widetilde{y_n} + \frac{h^3}{12}\widetilde{\lambda}^3\widetilde{y_n} + \frac{h^4}{24}\widetilde{\lambda}^4\widetilde{y_n}\Bigg].\end{aligned}$$

Dans l'exemple 5.5, les sommations sont parenthésées de la droite vers la gauche. Il en est de même pour les produits : ainsi, nous avons par exemple $\frac{h^4}{24}\widetilde{\lambda}^4 = (h \otimes (h \otimes (h \otimes (h \otimes \frac{1}{24} \otimes (\widetilde{\lambda} \otimes (\widetilde{\lambda} \otimes (\widetilde{\lambda} \otimes \widetilde{\lambda}))))))$.

L'analyse d'erreurs d'arrondi que nous proposons dans la suite du chapitre est générique et s'applique à n'importe quelle implémentation des méthodes de Runge-Kutta. Néanmoins, en guise d'exemple, nous instancierons la méthodologie dans le cas particulier de ces implémentations naïves.

5.2 Méthodologie

Dans cette section, nous présentons les différentes étapes de la méthodologie développée pour l'analyse des erreurs d'arrondi des méthodes de Runge-Kutta.

Considérons une méthode de Runge-Kutta appliquée à un système linéaire unidimensionnel, caractérisée par la relation suivante :

$$y_{n+1} = R(h, \lambda)y_n = \sum_{i=0}^{K} \alpha_i y_n$$

avec α_i dépendant à la fois de l'équation différentielle et du pas d'intégration de la méthode. Dans les exemples de la section 5.1.1, les coefficients α_i sont, par exemple, $h\lambda$, $\frac{h^2\lambda^2}{6}$ ou encore $\frac{h^4\lambda^4}{24}$.

L'implémentation naïve correspondante en arithmétique à virgule flottante (voir section 5.1.2) est de la forme :

$$\widetilde{y_{n+1}} = \widetilde{R}(h, \widetilde{\lambda}, \widetilde{y_n}) = \oplus_{i=0}^{K} \widetilde{\alpha_i} \otimes \widetilde{y_n}.$$

Les coefficients $\widetilde{\alpha_i}$ correspondant aux coefficients α_i donnés ci-dessus sont $h \otimes \widetilde{\lambda}$, $h \otimes h \oslash 6 \otimes \widetilde{\lambda} \otimes \widetilde{\lambda}$ et $h \otimes h \otimes h \otimes h \oslash 24 \otimes \widetilde{\lambda} \otimes \widetilde{\lambda} \otimes \widetilde{\lambda} \otimes \widetilde{\lambda}$.

Nous distinguons les erreurs d'arrondi dites locales et globales. L'erreur que nous souhaitons borner est celle qui s'accumule au fil des itérations de la méthode. Cette erreur est appelée erreur globale, est notée E_n (après n itérations) et est définie comme :

$$E_n = \widetilde{y_n} - y_n = \widetilde{R}(h, \widetilde{\lambda}, \widetilde{y_n}) - R(h, \lambda) y_n. \tag{5.6}$$

Pour pouvoir analyser les erreurs globales, nous définissons le concept d'erreur locale, qui correspond à l'erreur d'arrondi commise au cours d'une seule itération donnée de la méthode, sans prendre en compte l'erreur commise aux itérations précédentes. L'erreur locale à l'itération n, notée ε_n, est définie comme :

$$\varepsilon_0 = |\widetilde{y_0} - y_0| \qquad \forall n \in \mathbb{N}^*, \varepsilon_n = |\widetilde{R}(h, \widetilde{\lambda}, \widetilde{y_{n-1}}) - R(h, \lambda)\widetilde{y_{n-1}}|. \tag{5.7}$$

La méthodologie utilisée pour borner l'erreur globale d'une méthode numérique s'articule en plusieurs étapes :

- calculer une borne sur l'erreur d'arrondi $E_0 = \varepsilon_0$ commise lors du calcul de la valeur initiale y_0. Cette étape est plus ou moins difficile, selon que y_0 soit une constante, *e.g.* $y_0 = 1$, ou le résultat d'un autre algorithme, *e.g.* l'évaluation d'une exponentielle.
- calculer une borne sur l'erreur absolue de chaque coefficient α_i, *i.e.* une borne sur $|\widetilde{\alpha_i} - \alpha_i|$. Cette partie du travail a été déléguée à l'outil Gappa (voir chapitre 2), qui utilise les hypothèses de stabilité linéaire pour raffiner les résultats.
- à partir des bornes d'erreur sur les termes α_i, calculer une borne sur l'erreur locale en appliquant mécaniquement K fois le lemme 5.4, un résultat technique taillé sur mesure pour cette tâche. Nous tenons compte des dépassements graduels de capacité inférieurs et obtenons un résultat de la forme $\forall n, \varepsilon_n \leqslant Cu|\widetilde{y_{n-1}}| + D\eta$ avec C et D des constantes de petite taille (pour rappel, $u = \frac{1}{2}\beta^{1-p}$ est l'unité d'arrondi et $\eta = \beta^{e_{\min}-p+1}$ est le plus petit nombre flottant dénormalisé). De plus, pour $|\widetilde{y_{n-1}}| \geqslant M$ avec M une constante bien choisie, nous prouvons que $\varepsilon_n \leqslant Cu|\widetilde{y_{n-1}}|$, ce qui correspond à l'absence de dépassement de capacité inférieur. Dans le cas des méthodes de Runge-Kutta, la valeur de M est difficile à obtenir et dépend des paramètres du problème. En section 5.4, nous construisons mécaniquement les valeurs de C, D, mais aussi M pour les méthodes d'Euler, de RK2 et de RK4.

- instancier le théorème 5.9 avec les constantes C et D obtenues précédemment : ce théorème permet de construire une borne sur l'erreur globale absolue des méthodes de Runge-Kutta à partir de bornes sur les erreurs locales des itérations précédentes, en tenant compte des éventuels dépassements de capacité inférieurs.
- les dépassements de capacité supérieurs peuvent être gérés à part de l'analyse d'erreurs. Il est en fait possible de montrer que, si $\widetilde{y_0}$ n'est ni un infini ni NaN, une borne raisonnable sur sa valeur suffit à se prémunir du risque de dépassement de capacité supérieur (voir section 5.6).

Raffinement des bornes d'erreur : En l'absence de dépassement de capacité inférieur, les bornes obtenues sont sensiblement meilleures que dans un précédent article que j'ai rédigé avec Boldo et Chapoutot en 2017 [29]. Cette amélioration provient de l'utilisation de bornes optimales d'erreurs relatives étudiées en détail par Jeannerod et Rump en 2016 [119]. Nous utilisons plus précisément les lemmes 2.4, 2.5 et 2.6 de ce manuscrit.

5.3 Décroissance des solutions calculées

Les méthodes considérées sont stables au sens de la stabilité linéaire (voir section 5.1.1). D'un point de vue mathématique, la propriété de stabilité va de pair avec la décroissance des solutions obtenues au cours des itérations. Néanmoins, lorsque les méthodes sont implémentées en précision finie, la décroissance des solutions calculées n'est pas évidente *a priori*. Nous pouvons toutefois montrer, lorsque la méthode est stable, que cette propriété est vraie sous certaines conditions. Ce résultat, très utile, permettra de simplifier les raisonnements d'analyse d'erreurs et de gérer les éventuels dépassements de capacité supérieurs.

Lemme 5.1. *Soit $C \geqslant 0, M > 0$. Supposons que :*

- $Cu + |R(h,\lambda)| \leqslant 1$ *;*
- $\forall n \in \mathbb{N}^*, |\widetilde{y_n}| \geqslant M \Rightarrow \varepsilon_n \leqslant Cu|\widetilde{y_{n-1}}|$.

Alors :

$$\forall n \in \mathbb{N}^*, |\widetilde{y_n}| \geqslant M \Rightarrow |\widetilde{y_{n-1}}| \geqslant |\widetilde{y_n}| \geqslant M.$$

Démonstration. Soit $n \in \mathbb{N}^*$. Supposons que $|\widetilde{y_n}| \geqslant M$. Nous avons :
$|\widetilde{y_n}| = |\widetilde{R}(h,\widetilde{\lambda},\widetilde{y_{n-1}})| = |\widetilde{R}(h,\widetilde{\lambda},\widetilde{y_{n-1}}) - R(h,\lambda)\widetilde{y_{n-1}} + R(h,\lambda)\widetilde{y_{n-1}}|$.

Par une inégalité triangulaire :

$$|\widetilde{y_n}| \leqslant |\widetilde{R}(h,\widetilde{\lambda},\widetilde{y_{n-1}}) - R(h,\lambda)\widetilde{y_{n-1}}| + |R(h,\lambda)||\widetilde{y_{n-1}}| = \varepsilon_n + |R(h,\lambda)||\widetilde{y_{n-1}}|.$$

Comme $|\widetilde{y_n}| \geqslant M$, $\varepsilon_n \leqslant Cu|\widetilde{y_{n-1}}|$.

De plus, $Cu + |R(h,\lambda)| \leqslant 1$, donc :

$$M \leqslant |\widetilde{y_n}| \leqslant (Cu + |R(h,\lambda)|)|\widetilde{y_{n-1}}| \leqslant |\widetilde{y_{n-1}}|.$$

□

Une induction très simple permet ensuite d'obtenir le résultat suivant :

Lemme 5.2. *Soit $C \geqslant 0, M > 0$. Supposons que :*
- $Cu + |R(h,\lambda)| \leqslant 1$;
- $\forall n \in \mathbb{N}^*, |\widetilde{y_n}| \geqslant M \Rightarrow \varepsilon_n \leqslant Cu|\widetilde{y_{n-1}}|$.

Alors : $\forall n \in \mathbb{N}, |\widetilde{y_n}| \geqslant M \Rightarrow |\widetilde{y_n}| \leqslant |\widetilde{y_{n-1}}| \leqslant \cdots \leqslant |\widetilde{y_1}| \leqslant |\widetilde{y_0}|$.

Dans la section 5.6 (théorème 5.14), le lemme 5.2 sera utilisé pour vérifier une condition qui permet de se prémunir des dépassements de capacité supérieurs.

5.4 Erreurs locales

Comme stipulé par la méthodologie adoptée (voir section 5.2), nous commençons tout d'abord par borner les erreurs locales des méthodes de Runge-Kutta. La section 5.4.1 est vouée à des résultats préliminaires d'arithmétique à virgule flottante qui s'appliquent à une large classe d'implémentations de méthodes et tiennent comptent d'éventuels dépassements de capacité inférieurs. Ensuite, en section 5.4.2, nous fournissons des bornes d'erreurs locales pour les implémentations naïves de méthodes classiques, à savoir les méthodes d'Euler, de Runge-Kutta 2 et de Runge-Kutta 4.

5.4.1 Résultats génériques pour les erreurs locales

Plutôt que de borner directement les erreurs d'arrondi locales induites par une méthode de Runge-Kutta spécifique, nous proposons des résultats génériques et techniques qui permettront de reconstruire presque mécaniquement les bornes d'erreurs locales.

Dans un premier temps, nous nous intéressons à la sommation de deux nombres flottants X_1 et X_2 correspondant à des multiples d'une même valeur y et qui sont entachés d'erreurs. Nous bornons l'erreur d'arrondi commise lors de ce calcul.

Lemme 5.3. *Soit $y \in \mathbb{R}$. Soit $C_1, C_2, D_1, D_2 \in \mathbb{R}_+$. Soit $\alpha_1, \alpha_2 \in \mathbb{R}$. Soit $X_1, X_2 \in \mathbb{F}$. Supposons que :*
- $|X_1 - \alpha_1 y| \leqslant C_1 u|y| + D_1\eta$;
- $|X_2 - \alpha_2 y| \leqslant C_2 u|y| + D_2\eta$.

Alors :
$$|X_1 \oplus X_2 - (\alpha_1 + \alpha_2)y|$$
$$\leqslant \left(C_1u + C_2u + \frac{u}{1+u}(|\alpha_1| + |\alpha_2| + C_1u + C_2u)\right)|y| + (1+u)(D_1 + D_2)\eta.$$

Démonstration. La preuve repose sur les propriétés de l'arithmétique à virgule flottante et l'application du lemme 2.6 (car l'addition est exacte en cas de dépassement de capacité inférieur [83], d'où l'absence de terme en η).

$$
\begin{aligned}
&| X_1 \oplus X_2 - (\alpha_1 + \alpha_2)y | \\
&\quad \leqslant |X_1 \oplus X_2 - (X_1 + X_2)| + |X_1 + X_2 - (\alpha_1 + \alpha_2)y| \\
&\quad \leqslant \frac{u}{1+u} \times |X_1 + X_2| + (C_1 u + C_2 u)|y| + (D_1 + D_2)\eta \\
&\quad \leqslant \frac{u}{1+u} \times ((C_1 u + |\alpha_1|)|y| + (C_2 u + |\alpha_2|)|y|) \\
&\qquad + (C_1 u + C_2 u)|y| + (D_1 + D_2)\left(1 + \frac{u}{1+u}\right)\eta \\
&\quad = \left((C_1 + C_2)u + \frac{u}{1+u} \times (|\alpha_1| + |\alpha_2| + (C_1 + C_2)u)\right)|y| + \eta \frac{1+2u}{1+u}(D_1 + D_2) \\
&\quad \leqslant \left(C_1 u + C_2 u + \frac{u}{1+u}(|\alpha_1| + |\alpha_2| + C_1 u + C_2 u)\right)|y| + (1+u)(D_1 + D_2)\eta.
\end{aligned}
$$

□

Nous nous intéressons ensuite au calcul de $X_1 \oplus (X_2 \otimes y)$ avec X_1 un nombre flottant correspondant au calcul d'un multiple de y, X_1 et X_2 étant entachés d'erreurs. Rappelons que $\xi = \beta^{e_{\min}}$ est le plus petit nombre flottant normalisé positif.

Lemme 5.4. *Soit $y, X_1, X_2 \in \mathbb{F}$. Soit $C_1, C_2, D_1, M, P_2 \in \mathbb{R}_+$. Soit $\alpha_1 \in \mathbb{R}, \alpha_2 \in \mathbb{R}^*$. Supposons que :*

- $|X_1 - \alpha_1 y| \leqslant C_1 u|y| + D_1\eta$;
- $|y] \geqslant M \Rightarrow |X_1 - \alpha_1 y| \leqslant C_1 u|y|$;
- $|X_2 - \alpha_2| \leqslant C_2 u$;
- $\left|\frac{X_2 - \alpha_2}{\alpha_2}\right| \leqslant P_2 < 1$;
- $M \geqslant \frac{\xi}{|\alpha_2|(1-P_2)}$.

Alors :

- $|y| \geqslant M \Rightarrow |X_1 \oplus (X_2 \otimes y) - (\alpha_1 + \alpha_2)y|$
  $\leqslant |y|\left(C_1 u + C_2 u + \frac{u}{1+u}(|\alpha_1| + 2|\alpha_2| + C_1 u + 2C_2 u) + \left(\frac{u}{1+u}\right)^2 (C_2 u + |\alpha_2|)\right)$;
- $|X_1 \oplus (X_2 \otimes y) - (\alpha_1 + \alpha_2)y|$
  $\leqslant |y|\left(C_1 u + C_2 u + \frac{u}{1+u}(|\alpha_1| + 2|\alpha_2| + C_1 u + 2C_2 u) + \left(\frac{u}{1+u}\right)^2 (C_2 u + |\alpha_2|)\right)$
  $+\eta(1+u)\left(\frac{1}{2} + D_1\right)$.

Démonstration. La preuve repose sur une application du lemme 5.3 avec les mêmes variables X_1, α_1 et α_2. En revanche, le paramètre X_2 du lemme 5.3 est ici égal à $X_2 \otimes y$. Les assertions requises sur X_1, α_1, C_1 et D_1 pour appliquer le lemme 5.3 sont directement déduites des hypothèses. Il reste à exhiber C_2 et D_2, ce qui correspond à borner l'erreur sur $X_2 \otimes y$:

- Supposons que $|y|$ est quelconque. Nous utilisons le lemme 2.4 et prouvons :

$$
\begin{aligned}
|X_2 \otimes y - \alpha_2 y| &\leqslant |X_2 \otimes y - X_2 \times y| + |y|\,|X_2 - \alpha_2| \\
&\leqslant \frac{u}{1+u}\,|X_2|\,|y| + \frac{\eta}{2} + |y|\,C_2 u \\
&\leqslant \frac{u}{1+u}\,(C_2 u + |\alpha_2|)\,|y| + |y|\,C_2 u + \frac{\eta}{2} \\
&= |y|\left(C_2 u + \frac{u}{1+u}(C_2 u + |\alpha_2|)\right) + \frac{\eta}{2}.
\end{aligned}
$$

Donc, nous pouvons prendre $\left(C_2 + \frac{1}{1+u}(C_2 u + |\alpha_2|)\right)$ comme valeur de C_2 (du lemme 5.3), puis $\frac{1}{2}$ comme valeur de D_2 (du lemme 5.3) et avons toutes les hypothèses nécessaires à l'application du lemme 5.3, qui permet d'obtenir :

$$
\begin{aligned}
&|X_1 \oplus X_2 \otimes y - (\alpha_1 + \alpha_2) y| \\
&\leqslant \Big(C_1 u + C_2 u + \frac{u}{1+u}(C_2 u + |\alpha_2|) + \frac{u}{1+u}(|\alpha_1| + |\alpha_2| + C_1 u + C_2 u \\
&\qquad + \frac{u}{1+u}(C_2 u + |\alpha_2|))\Big)\,|y| + \eta(1+u)\left(\frac{1}{2} + D_1\right) = \\
&\left((C_1 + C_2)u + \frac{u}{1+u}\,(|\alpha_1| + 2|\alpha_2| + (C_1 + 2C_2)u) + \left(\frac{u}{1+u}\right)^2 (C_2 u + |\alpha_2|)\right)|y| \\
&\qquad + \eta(1+u)\left(\frac{1}{2} + D_1\right).
\end{aligned}
$$

• Supposons que $|y| \geqslant M$.

Nous savons que $|y| \geqslant \frac{\xi}{|\alpha_2|(1-P_2)}$. De plus, $\left|\frac{X_2 - \alpha_2}{\alpha_2}\right| \leqslant P_2$ *i.e.* $|X_2 - \alpha_2| \leqslant P_2|\alpha_2|$ et donc :

$$|\alpha_2| - |X_2| \leqslant P_2|\alpha_2| \text{ } i.e \text{ } |X_2| \geqslant (1 - P_2)|\alpha_2|$$

et comme $P_2 < 1$, nous avons :

$$|X_2 y| \geqslant (1 - P_2)|\alpha_2| \frac{\xi}{|\alpha_2|(1 - P_2)} = \xi.$$

Donc $|X_2 \otimes y - X_2 y| \leqslant \frac{u}{1+u}\,|X_2|\,|y|$ parce que la multiplication de X_2 par y ne produit pas de dépassement de capacité inférieur car $|X_2 y| \geqslant \xi$ (voir lemme 2.5).

Nous appliquons le même raisonnement que dans le premier cas avec $\eta = 0$.

□

Les hypothèses peuvent sembler redondantes, en particulier la borne P_2 sur l'erreur relative commise sur α_2 et la borne C_2 sur l'erreur absolue commise sur ce même coefficient. Il semble en effet possible d'obtenir C_2 à partir de P_2. Cependant, dans le cadre d'application de ce lemme, α_2 n'est pas une constante dont on connaît précisément la valeur. La seule information sur α_2 consiste en un intervalle de valeurs possibles et nous souhaitons obtenir des bornes d'erreurs valides quelle que soit la valeur prise dans cet intervalle. Cet intervalle de valeurs est en effet déduit des hypothèses de stabilité sur $h\lambda$. Par exemple, pour la méthode RK4, pour $\alpha_2 = \frac{h^2\lambda^2}{6}$ et $X_2 = \circ\left[h^2 \frac{1}{6} \widetilde{\lambda}^2\right]$, comme la région de stabilité de $h\lambda$ fournit l'hypothèse $-2 \leqslant h\lambda \leqslant 0$, la seule information sur α_2 est $0 \leqslant \alpha_2 \leqslant \frac{2}{3}$. À partir de ce seul intervalle, nous devons être en mesure de déduire des bornes P_2 et C_2 qui soient décorrélées des valeurs exactes de h et λ et donc de α_2.

Ces résultats sont taillés sur mesure pour borner les erreurs d'arrondi locales des méthodes de Runge-Kutta lorsque les dépassements graduels de capacité inférieurs sont pris en compte (voir section 5.4.2). Dans le cas général, nous notons l'apparition d'un terme en η correspondant à la contribution d'éventuels dépassements de capacité inférieurs. Au contraire, lorsque $|y|$ est plus grand que le seuil M, il est possible d'utiliser la seconde hypothèse du lemme 5.4 et d'obtenir une borne d'erreur plus fine.

5.4.2 Bornes sur l'erreur locale de méthodes de Runge-Kutta

Dans cette section, les résultats génériques de la section 5.4.1 sont utilisés dans le but de borner les erreurs relatives locales de méthodes de Runge-Kutta classiques. Cela implique de borner chaque terme du polynôme $R(h,\lambda)$ (*e.g.* $\frac{h\lambda}{6}, \frac{h^2\lambda^2}{2}, \frac{h^3\lambda^3}{12}$) ainsi que les erreurs d'arrondi associées.

Pour assurer un plus haut niveau de confiance dans nos résultats, nous utilisons l'outil Gappa [57, 61] (voir section 2.2.4) pour borner l'erreur d'arrondi associée à ces coefficients. Nous vérifions toutefois manuellement si les résultats obtenus par Gappa peuvent être raffinés en raisonnant plus finement sur la forme des expressions et prêtons plus particulièrement attention aux cas où l'outil n'est pas en mesure de donner une borne « optimale ». Néanmoins, les résultats obtenus par Gappa seront retenus dans l'énoncé des lemmes qui suivent. Les scripts Gappa constituent l'annexe B du manuscrit.

Dans l'ensemble de la section, nous supposerons que les coefficients de la forme $\frac{h^k\lambda^k}{p}$ sont évalués dans l'ordre qui suit :

$$h \otimes \cdots \otimes h \oslash p \otimes \widetilde{\lambda} \otimes \cdots \otimes \widetilde{\lambda}.$$

Nous avons néanmoins testé d'autres ordres d'évaluation et les bornes exhibées semblent demeurer valides.

Dans cette section, le format considéré est *binary64*. Les méthodes considérées sont stables et nous exploitons la région de stabilité du produit $h\lambda$.

Remarque 3. *Les bornes que nous proposons ou souhaitons vérifier via Gappa tirent parti de la forme des expressions, notamment du nombre d'arrondi qu'elles comportent :*

- *concernant l'erreur relative :*
  - *pour une expression comportant n arrondis (i.e. les opérations arrondies $\otimes$ et $\oslash$, les coefficients $\widetilde{\lambda}$, etc), l'erreur relative est bornée par $nu+O(u^2)$. Lorsque la constante associée au grand O est suffisamment petite, nous pourrons retenir la borne $(n+0{,}01)u$* [107, §3.4]*,*
  - *cependant, cette borne peut dans certains cas être raffinée : lorsque $n \leqslant 3$, Jeannerod et Rump* [119, §5.1] *montrent que le terme en u^2 peut être supprimé de la borne et que l'erreur relative peut être bornée par nu,*
- *concernant l'erreur absolue :*
  - *pour une expression donnée $f(h,\lambda)$ comportant n arrondis, l'erreur absolue est bornée par $(n+0{,}01)u|f(h,\lambda)|$, la borne sur $|f(h,\lambda)|$ étant obtenue à partir des contraintes sur $h\lambda$ imposées par la région de stabilité* [107, §3.4]*,*

- *dans certains cas, e.g. dans certaines expressions uniquement constituées d'additions ou de multiplications, nous parvenons à obtenir la borne* $nu|f(h,\lambda)|$ *sans terme en* u^2 [89].

La première méthode étudiée est la méthode d'Euler :

Lemme 5.5. Borne sur l'erreur locale pour la méthode d'Euler
Soit $h \in \mathbb{F}$, $\lambda \in \mathbb{R}$. *Soit* $n \in \mathbb{N}$. *Soit* $(\widetilde{y_n})_{n\in\mathbb{N}}$ *la suite des solutions calculées via une implémentation « naïve » de la méthode d'Euler (voir section 5.1.2) dans le format binary64. Soit* $(\varepsilon_n)_{n\in\mathbb{N}}$ *la suite des erreurs locales associées à cette implémentation. Supposons que* $-2 \leqslant h\lambda \leqslant -2^{-100}$ *(stabilité) et* $2^{-60} \leqslant h \leqslant 1$. *Alors :*

- $|\widetilde{y_n}| \geqslant \frac{\xi}{2(1-2u)} \Rightarrow \varepsilon_{n+1} \leqslant 9{,}01u\,|\widetilde{y_n}|$ *;*
- $\varepsilon_{n+1} \leqslant 9{,}01u\,|\widetilde{y_n}| + 1{,}01\eta$.

Démonstration. La démonstration repose sur l'application du lemme 5.4 avec $y = \widetilde{y_n}$, $X_1 = \widetilde{y_n}$, $\alpha_1 = 1$, $C_1 = 0$, $D_1 = 0$, $X_2 = h \otimes \widetilde{\lambda}$ et $\alpha_2 = h\lambda$. Il faut exhiber la constante C_2.

En utilisant Gappa (voir annexe B, fichier `euler.gappa`), nous obtenons $|X_2 - h\lambda| \leqslant 4 \times 2^{-53}$ à partir des intervalles de valeurs que peuvent prendre $h\lambda$ et h, *i.e.* $C_2 = 4$. Comme il y a 2 arrondis (la multiplication et l'arrondi sur λ), cela est cohérent avec la borne $2u|h\lambda| \leqslant 2u \times 2$ (voir remarque 3).

Concernant la borne sur l'erreur relative, d'après la remarque 3, comme le nombre d'arrondis est $2 \leqslant 3$, nous pensons que la borne la plus fine pouvant être obtenue est $\left|\frac{X_2-h\lambda}{h\lambda}\right| \leqslant 2\times 2^{-53}$. Gappa ne parvient pas à vérifier cette borne mais vérifie $\left|\frac{X_2-h\lambda}{h\lambda}\right| \leqslant 2{,}01\times 2^{-53}$. En fait, les théorèmes utilisés par Gappa utilise les bornes d'erreurs standards en u et non les bornes raffinées faisant intervenir le terme $\frac{u}{1+u}$. Le remplacement de la quantité u par $\frac{u}{1+u}$ dans ces théorèmes semble suffisante pour démontrer automatiquement la borne plus fine $P_2 = 2u$, que nous choisissons de retenir.

L'application du lemme 5.4 donne ensuite

$$|\widetilde{y_n}| \geqslant \frac{\xi}{2(1-2u)} \Rightarrow \varepsilon_{n+1} \leqslant |\widetilde{y_n}|\left(4u + u\left(1 + 2|h\lambda| + 8u\right) + u^2(4u + |h\lambda|)\right).$$

Il faut noter que $\frac{\xi}{2(1-2u)}$ a été choisi de manière à garantir les conditions du lemme 5.4 sur le seuil M. De plus, le lemme 5.4 nous donne :

$$\varepsilon_{n+1} \leqslant |\widetilde{y_n}|\left(4u + u\left(1 + 2|h\lambda| + 8u\right) + u^2(4u + |h\lambda|)\right) + 0{,}5(1+u)\eta.$$

Comme $|h\lambda| \leqslant 2$, nous avons :

$$\varepsilon_{n+1} \leqslant |\widetilde{y_n}|\left(9u + 10u^2 + 4u^3\right) + 0{,}5(1+u)\eta \leqslant 9{,}01u\,|\widetilde{y_n}| + 1{,}01\eta$$

et

$$|\widetilde{y_n}| \geqslant \frac{\xi}{2(1-2{,}01u)} \Rightarrow \varepsilon_{n+1} \leqslant |\widetilde{y_n}|\left(9u + 10u^2 + 4u^3\right) \leqslant 9{,}01u\,|\widetilde{y_n}|.$$

□

Lemme 5.6. Borne sur l'erreur locale pour la méthode RK2
Soit $h \in \mathbb{F}$, $\lambda \in \mathbb{R}$. Soit $n \in \mathbb{N}$. Soit $(\widetilde{y_n})_{n\in\mathbb{N}}$ la suite des solutions calculées via une implémentation « naïve » de la méthode RK2 (voir section 5.1.2) dans le format binary64. Soit $(\varepsilon_n)_{n\in\mathbb{N}}$ la suite des erreurs locales associées à cette implémentation. Supposons que $-2 \leqslant h\lambda \leqslant -2^{-100}$ (stabilité) et $2^{-60} \leqslant h \leqslant 1$. Alors :

- $|\widetilde{y_n}| \geqslant \frac{\xi}{2(1-5{,}01u)} \Rightarrow \varepsilon_{n+1} \leqslant 24{,}03u\,|\widetilde{y_n}|$;
- $\varepsilon_{n+1} \leqslant 24{,}03u\,|\widetilde{y_n}| + 1{,}01\eta$.

Démonstration. L'implémentation naïve de RK2 est caractérisée par la relation :

$$\widetilde{y_{n+1}} = \circ\left[\widetilde{y_n} + h\widetilde{\lambda}\widetilde{y_n} + hh\tfrac{1}{2}\widetilde{\lambda}\widetilde{\lambda}\widetilde{y_n}\right].$$

L'hypothèse de stabilité linéaire est $\left|1 + h\lambda + \frac{h^2\lambda^2}{2}\right| \leqslant 1$. En prenant comme modèle la démonstration du lemme 5.5 pour la méthode d'Euler, nous appliquons plusieurs fois le lemme 5.4 pour reconstruire la borne sur l'erreur locale de la méthode RK2. Le tableau 5.1 contient les valeurs successives correspondant à α, C, P_2 et D. Le seuil M à partir duquel il n'y a pas de dépassement de capacité inférieur au cours d'un pas de calcul est déterminé par les différents seuils M_{loc} correspondant à chaque terme de ce pas de calcul : il suffit de choisir M comme la valeur maximale prise par M_{loc}.

Les 3 premières lignes du tableau 5.1 réutilisent les valeurs trouvées pour borner l'erreur locale de la méthode d'Euler dans le lemme 5.5 (car l'hypothèse sur $h\lambda$ est identique, *i.e.* $-2 \leqslant h\lambda \leqslant -2^{-100}$).

Concernant la borne de la ligne 4, Gappa (voir annexe B, fichier `rk2.gappa`) donne $C_2 = 10{,}01$ et $P_2 = 5{,}01u$. Ces résultats sont cohérents avec la remarque 3, car il n'y a pas 6 arrondis, mais seulement 5, la division par 2 étant exacte.

La dernière ligne (ligne 5) correspond à l'application du lemme 5.4 avec les valeurs C et α des lignes 3 et 4 et l'hypothèse $-2 \leqslant h\lambda \leqslant 0$:

$$\begin{aligned}\varepsilon_{n+1} &\leqslant |\widetilde{y_n}|\left(9{,}01u + 10{,}01u + u\left(|1 + h\lambda| + 2\left|\frac{h^2\lambda^2}{2}\right|\right) + 0{,}01u\right) + (1+u)(0{,}5 + 0{,}5(1+u))\eta \\ &\leqslant 24{,}03u\,|\widetilde{y_n}| + 1{,}01\eta.\end{aligned}$$

Ici, M est la valeur maximale prise par M_{loc}, *i.e.*, $\frac{\xi}{2(1-5{,}01u)}$.

Tableau 5.1 – Étapes pour borner l'erreur locale de la méthode RK2

Ligne	Terme FP	C	P_2	M_{loc}	D
1	$\widetilde{y_n}$	0	–	0	0
2	$h \otimes \widetilde{\lambda}$	4	$2u$	–	–
3	$\circ\left[\widetilde{y_n} + h\widetilde{\lambda}\widetilde{y_n}\right]$	$9{,}01u$	–	$\frac{\xi}{2(1-2{,}01u)}$	$0{,}5(1+u)$
4	$\circ\left[hh\frac{1}{2}\widetilde{\lambda}\widetilde{\lambda}\right]$	$10{,}01u$	$5{,}01u$	–	–
5	$\circ[\widetilde{y_n} + \dots]$	24,03	–	$\frac{\xi}{2(1-5{,}01u)}$	$1 + 1{,}51u$

□

Lemme 5.7. Borne sur l'erreur locale pour la méthode RK4
Soit $h \in \mathbb{F}$, $\lambda \in \mathbb{R}$. Soit $n \in \mathbb{N}$. Soit $(\widetilde{y_n})_{n\in\mathbb{N}}$ la suite des solutions calculées via une implémentation « naïve » de la méthode RK4 (voir section 5.1.2) dans le format binary64. Soit $(\varepsilon_n)_{n\in\mathbb{N}}$ la suite des erreurs locales associées à cette implémentation. Supposons que $-2 \leqslant h\lambda \leqslant -2^{-100}$ (stabilité) et $2^{-60} \leqslant h \leqslant 1$. Alors :

- $|\widetilde{y_n}| \geqslant \frac{3\xi}{1-3{,}01u} \Rightarrow \varepsilon_{n+1} \leqslant 54{,}47\,|\widetilde{y_n}|$;
- $\varepsilon_{n+1} \leqslant 54{,}47u\,|\widetilde{y_n}| + 5{,}34\eta$.

Démonstration. L'hypothèse de stabilité linéaire associée à la méthode RK4 est :

$$\left|1 + h\lambda + \frac{h^2\lambda^2}{2} + \frac{h^3\lambda^3}{12} + \frac{h^4\lambda^4}{24}\right| \leqslant 1,$$

Une condition optimale de stabilité serait de la forme $-2{,}785\,293\,563\,405\,281\,7... \leqslant h\lambda \leqslant 0$. Nous choisissons néanmoins, au prix de bornes moins optimales, de nous limiter à la condition $-2 \leqslant h\lambda \leqslant 0$. Ainsi, nous travaillons sur le même domaine de valeurs que pour les méthodes d'Euler et de RK2, ce qui nous permet de comparer les résultats obtenus. Comme pour Euler et RK2, nous procédons à des applications successives du lemme 5.4 et utilisons Gappa (voir annexe B, fichier `RK4.gappa`) pour borner les coefficients en $h\lambda$ *via* les intervalles de valeurs pour $h\lambda$ et h. La tableau 5.2 contient les valeurs correspondant à α, C, P_2, D et M_{loc} pour les termes d'un pas de calcul. Ici, nous pouvons prendre $M = \frac{3\xi}{1-3{,}01u}$, qui est la valeur maximale prise par M_{loc}. Notons que, dans ce cas, M n'est pas la valeur M_{loc} correspondant au dernier terme du pas de calcul.

Les résultats du tableau 5.2 sont obtenus de la manière suivante :

- ligne 1 : trivialement, $C = D = M_{loc} = 0$.
- ligne 2 : Gappa obtient $\left|\circ\left[h\frac{1}{6}\widetilde{\lambda}\right] - \frac{h\lambda}{6}\right| \leqslant u$ et $\left|\frac{\circ\left[h\frac{1}{6}\widetilde{\lambda}\right]-\frac{h\lambda}{6}}{\frac{h\lambda}{6}}\right| \leqslant 3{,}01u$, ce qui est cohérent avec le fait qu'il y ait 3 arrondis et que $\left|\frac{h\lambda}{6}\right| \leqslant \frac{1}{3}$. Nous pensons cependant que la borne d'erreur relative peut être réduite à $3u$ (voir [119, §5.1] et remarque 3).
- ligne 3 : par application du lemme 5.4 :

  $$\varepsilon_{n+1} \leqslant |\widetilde{y_n}|\left(0u + u + u\left(1 + 2\left|\frac{h\lambda}{6}\right|\right) + 0{,}01u\right) + 0{,}5(1+u) \leqslant 2{,}68u\,|\widetilde{y_n}| + 0{,}5(1+u).$$

  Nous choisissons $M_{loc} = \frac{3\xi}{1-3{,}01u}$.
- ligne 4 : Gappa obtient $\left|\circ\left[h\frac{1}{3}\widetilde{\lambda}\right] - \frac{h\lambda}{3}\right| \leqslant 2u$ et $\left|\frac{\circ\left[h\frac{1}{3}\widetilde{\lambda}\right]-\frac{h\lambda}{3}}{\frac{h\lambda}{3}}\right| \leqslant 3{,}01u$, ce qui est cohérent avec le fait qu'il y ait 3 arrondis et que $\left|\frac{h\lambda}{6}\right| \leqslant \frac{1}{3}$. Nous pensons cependant que la borne d'erreur relative peut être réduite à $3u$ (voir [119, §5.1] et remarque 3).

Tableau 5.2 – Étapes pour borner l'erreur locale de la méthode RK4

Ligne	Terme FP	C	P_2	M_{loc}	D
1	$\widetilde{y_n}$	0	–	0	0
2	$h \otimes \frac{1}{6} \otimes \widetilde{\lambda}$	1	$3{,}01u$	–	–
3	$\circ\left[\widetilde{y_n} + h\frac{1}{6}\widetilde{\lambda}\right]$	2,68	–	$\frac{3\xi}{1-3{,}01u}$	$0{,}5(1+u)$
4	$h \otimes \frac{1}{3} \otimes \widetilde{\lambda}$	2	$3{,}01u$	–	–
5	$\circ\left[\widetilde{y_n} + h\frac{1}{6}\widetilde{\lambda} + h\frac{1}{3}\widetilde{\lambda}\right]$	6,69	–	$\frac{3\xi}{2(1-3{,}01u)}$	$1 + 1{,}51u$
6	$\circ\left[h^2\frac{1}{6}\widetilde{\lambda}^2\right]$	4	$6{,}01u$	–	–
7	$\circ\left[\widetilde{y_n} + \cdots + h^2\frac{1}{6}\widetilde{\lambda}^2\right]$	12,04	–	$\frac{3\xi}{2(1-6{,}01u)}$	$1{,}53(1+u)$
8	$h \otimes \frac{1}{3} \otimes \widetilde{\lambda}$	2	$3{,}01u$	–	–
9	$\circ\left[\widetilde{y_n} + \cdots + h\frac{1}{3}\widetilde{\lambda}\right]$	16,05	–	$\frac{3\xi}{2(1-3{,}01u)}$	$2{,}06 + 1{,}55u$
10	$\circ\left[h^2\frac{1}{6}\widetilde{\lambda}^2\right]$	4	$6{,}01u$	–	–
11	$\circ\left[\widetilde{y_n} + \cdots + h^2\frac{1}{6}\widetilde{\lambda}^2\right]$	21,4	–	$\frac{3\xi}{2(1-6{,}01u)}$	$2{,}59 + 1{,}57u$
12	$\circ\left[h^3\frac{1}{12}\widetilde{\lambda}^3\right]$	6	$9{,}01u$	–	–
13	$\circ\left[\widetilde{y_n} + \cdots + h^3\frac{1}{12}\widetilde{\lambda}^3\right]$	29,41	–	$\frac{3\xi}{2(1-9{,}01u)}$	$3{,}13 + 1{,}59u$
14	$h \otimes \frac{1}{6} \otimes \widetilde{\lambda}$	1	$3{,}01u$	–	–
15	$\circ\left[\widetilde{y_n} + \cdots + h\frac{1}{6}\widetilde{\lambda}\right]$	31,42	–	$\frac{3\xi}{1-3{,}01u}$	$3{,}67 + 1{,}61u$
16	$\circ\left[h^2\frac{1}{6}\widetilde{\lambda}^2\right]$	4	$6{,}01u$	–	–
17	$\circ\left[\widetilde{y_n} + \cdots + h^2\frac{1}{6}\widetilde{\lambda}^2\right]$	37,1	–	$\frac{3\xi}{2(1-6{,}01u)}$	$4{,}22 + 1{,}63u$
18	$\circ\left[h^3\frac{1}{12}\widetilde{\lambda}^3\right]$	6	$9{,}01u$	–	–
19	$\circ\left[\widetilde{y_n} + \cdots + h^3\frac{1}{12}\widetilde{\lambda}^3\right]$	44,78	–	$\frac{3\xi}{2(1-9{,}01u)}$	$4{,}77 + 1{,}65u$
20	$\circ\left[h^4\frac{1}{24}\widetilde{\lambda}^4\right]$	8	$12{,}01u$	–	–
21	$\circ\left[\widetilde{y_n} + \cdots + h^4\frac{1}{24}\widetilde{\lambda}^4\right]$	54,47	–	$\frac{3\xi}{2(1-12{,}01u)}$	$5{,}33 + 1{,}67u$

- ligne 5 : par application du lemme 5.4 :

$$\begin{aligned}\varepsilon_{n+1} &\leqslant |\widetilde{y_n}|\left(2{,}68u + 2u + u\left(\left|1 + \frac{h\lambda}{6}\right| + 2\left|\frac{h\lambda}{3}\right|\right) + 0{,}01u\right) + (1+u)(0{,}5(2+u))\eta \\ &\leqslant 6{,}69u\,|\widetilde{y_n}| + (1 + 1{,}51u)\eta.\end{aligned}$$

  Nous choisissons $M_{loc} = \frac{3\xi}{2(1-3{,}01u)}$.
- ligne 6 : Gappa obtient $\left|\circ\left[h^2\frac{1}{6}\widetilde{\lambda}^2\right] - \frac{h^2\lambda^2}{6}\right| \leqslant 4u$ et $\left|\frac{\circ\left[h^2\frac{1}{6}\widetilde{\lambda}^2\right] - \frac{h^2\lambda^2}{6}}{\frac{h^2\lambda^2}{6}}\right| \leqslant 6{,}01u$, ce qui est cohérent avec la remarque 3 car il y a 6 arrondis et que $\left|\frac{h^2\lambda^2}{6}\right| \leqslant \frac{2}{3}$.
- ligne 7 : par application du lemme 5.4 :

$$\begin{aligned}\varepsilon_{n+1} &\leqslant |\widetilde{y_n}|\left(6{,}69u + 4u + u\left(\left|1 + \frac{h\lambda}{6} + \frac{h\lambda}{3}\right| + 2\left|\frac{h^2\lambda^2}{6}\right|\right) + 0{,}01u\right)\\ &\quad + (1+u)(0{,}5 + 1 + 1{,}51u)\eta\\ &\leqslant 12{,}04u\,|\widetilde{y_n}| + 1{,}53(1+u)\eta.\end{aligned}$$

Nous choisissons $M_{loc} = \frac{3\xi}{2(1-6{,}01u)}$.

- ligne 8 : voir ligne 4.
- ligne 9 : par application du lemme 5.4 :

$$\begin{aligned}\varepsilon_{n+1} &\leqslant |\widetilde{y_n}|\left(12{,}04u + 2u + u\left(\left|1 + \frac{h\lambda}{6} + \frac{h\lambda}{3} + \frac{h^2\lambda^2}{6}\right| + 2\left|\frac{h\lambda}{3}\right|\right) + 0{,}01u\right)\\ &\quad + (1+u)(0{,}5 + 1{,}53(1+u))\eta\\ &\leqslant 16{,}05u\,|\widetilde{y_n}| + (2{,}06 + 1{,}55u)\eta.\end{aligned}$$

Nous choisissons $M_{loc} = \frac{3\xi}{2(1-3{,}01u)}$.

- ligne 10 : voir ligne 6.
- ligne 11 : par application du lemme 5.4 :

$$\begin{aligned}\varepsilon_{n+1} &\leqslant |\widetilde{y_n}|\left(16{,}05u + 4u + u\left(\left|1 + \frac{h\lambda}{6} + \frac{h\lambda}{3} + \frac{h^2\lambda^2}{6} + \frac{h\lambda}{3}\right| + 2\left|\frac{h^2\lambda^2}{6}\right|\right) + 0{,}01u\right)\\ &\quad + (1+u)(0{,}5 + 2{,}06 + 1{,}55u)\eta\\ &\leqslant 21{,}4u\,|\widetilde{y_n}| + (2{,}59 + 1{,}57u)\eta.\end{aligned}$$

Nous choisissons $M_{loc} = \frac{3\xi}{2(1-6{,}01u)}$.

- ligne 12 : Gappa obtient $\left|\circ\left[h^3\frac{1}{12}\widetilde{\lambda}^3\right] - \frac{h^3\lambda^3}{12}\right| \leqslant 6u$ et $\left|\frac{\circ\left[h^3\frac{1}{12}\widetilde{\lambda}^3\right] - \frac{h^3\lambda^3}{12}}{\frac{h^3\lambda^3}{12}}\right| \leqslant 9{,}01u$, ce qui est cohérent avec la remarque 3 car il y a 9 arrondis et que $\left|\frac{h^3\lambda^3}{12}\right| \leqslant \frac{2}{3}$.

- ligne 13 : par application du lemme 5.4 :

$$\begin{aligned}&\varepsilon_{n+1} \leqslant\\ &|\widetilde{y_n}|\left(21{,}4u + 6u + u\left(\left|1 + \frac{h\lambda}{6} + \frac{h\lambda}{3} + \frac{h^2\lambda^2}{6} + \frac{h\lambda}{3} + \frac{h^2\lambda^2}{6}\right| + 2\left|\frac{h^3\lambda^3}{12}\right|\right) + 0{,}01u\right)\\ &\quad + (1+u)(0{,}5 + 2{,}59 + 1{,}57u)\eta\\ &\leqslant 29{,}41u\,|\widetilde{y_n}| + (3{,}13 + 1{,}59u)\eta.\end{aligned}$$

Nous choisissons $M_{loc} = \frac{3\xi}{2(1-9{,}01u)}$.

- ligne 14 : voir ligne 1.
- ligne 15 : par application du lemme 5.4 :

$$\begin{aligned}
&\varepsilon_{n+1} \leqslant \\
&|\widetilde{y_n}|\left(29{,}41u + u + u\left(\left|1 + \frac{h\lambda}{6} + \frac{h\lambda}{3} + \frac{h^2\lambda^2}{6} + \frac{h\lambda}{3} + \frac{h^2\lambda^2}{6} + \frac{h^3\lambda^3}{12}\right| + 2\left|\frac{h\lambda}{6}\right|\right) + 0{,}01u\right) \\
&\quad + (1+u)(0{,}5 + 3{,}13 + 1{,}59u)\eta \\
&\leqslant 31{,}42u\,|\widetilde{y_n}| + (3{,}67 + 1{,}61u)\eta.
\end{aligned}$$

Nous choisissons $M_{loc} = \frac{3\xi}{1-3{,}01u)}$.

- ligne 16 : voir ligne 6.
- ligne 17 : par application du lemme 5.4 :

$$\begin{aligned}
\varepsilon_{n+1} &\leqslant |\widetilde{y_n}|\left(31{,}42u + 4u + u\left(\left|1 + \frac{2h\lambda}{6} + \frac{2h\lambda}{3} + \frac{2h^2\lambda^2}{6} + \frac{h^3\lambda^3}{12}\right| + 2\left|\frac{h^2\lambda^2}{6}\right|\right) + 0{,}01u\right) \\
&\quad + (1+u)(0{,}5 + 3{,}67 + 1{,}61u)\eta \\
&\leqslant 37{,}1u\,|\widetilde{y_n}| + (4{,}22 + 1{,}63u)\eta.
\end{aligned}$$

Nous choisissons $M_{loc} = \frac{3\xi}{2(1-6{,}01u)}$.

- ligne 18 : voir ligne 12.
- ligne 19 : par application du lemme 5.4 :

$$\begin{aligned}
\varepsilon_{n+1} &\leqslant |\widetilde{y_n}|\left(37{,}1u + 6u + u\left(\left|1 + \frac{2h\lambda}{6} + \frac{2h\lambda}{3} + \frac{3h^2\lambda^2}{6} + \frac{h^3\lambda^3}{12}\right| + 2\left|\frac{h^3\lambda^3}{12}\right|\right) + 0{,}01u\right) \\
&\quad + (1+u)(0{,}5 + 4{,}22 + 1{,}63u)\eta \\
&\leqslant 44{,}78u\,|\widetilde{y_n}| + (4{,}77 + 1{,}65u)\eta.
\end{aligned}$$

Nous choisissons $M_{loc} = \frac{3\xi}{2(1-9{,}01u)}$.

- ligne 20 : Gappa obtient $\left|\circ\left[h^4\frac{1}{24}\widetilde{\lambda}^4\right] - \frac{h^4\lambda^4}{24}\right| \leqslant 8u$ et $\left|\frac{\circ\left[h^4\frac{1}{24}\widetilde{\lambda}^4\right] - \frac{h^4\lambda^4}{24}}{\frac{h^4\lambda^4}{24}}\right| \leqslant 12{,}01u$, ce qui est cohérent avec la remarque 3 car il y a 12 arrondis et que $\left|\frac{h^4\lambda^4}{24}\right| \leqslant \frac{2}{3}$.

- ligne 21 : par application du lemme 5.4 :

$$\begin{aligned}
&\varepsilon_{n+1} \leqslant \\
&|\widetilde{y_n}|\left(44{,}78u + 8u + u\left(\left|1 + \frac{2h\lambda}{6} + \frac{2h\lambda}{3} + \frac{3h^2\lambda^2}{6} + \frac{2h^3\lambda^3}{12}\right| + 2\left|\frac{h^4\lambda^4}{24}\right|\right) + 0{,}01u\right) \\
&\quad + (1+u)(0{,}5 + 4{,}77 + 1{,}65u)\eta \\
&\leqslant 54{,}47u\,|\widetilde{y_n}| + 5{,}34\eta.
\end{aligned}$$

Nous choisissons $M_{loc} = \frac{3\xi}{2(1-12{,}01u)}$.

□

Le tableau 5.3 synthétise les valeurs finales de C, D et M pour les implémentations naïves des méthodes d'Euler, de RK2 et de RK4 en *binary64*.

Tableau 5.3 – Synthèse des constantes d'erreurs locales pour Euler, RK2 et RK4

Méthode	C	D	M	résultat
Euler	$9{,}01u$	1,01	$\frac{\xi}{2(1-2u)}$	lemme 5.5
RK2	$24{,}03u$	1,01	$\frac{\xi}{2(1-5{,}01u)}$	lemme 5.6
RK4	$54{,}47u$	5,34	$\frac{3\xi}{1-3u}$	lemme 5.7

5.5 Erreurs globales

En section 5.4, nous avons exhibé des bornes sur l'erreur locale de méthodes classiques. Comme indiqué en section 5.2, l'étape suivante consiste à borner l'erreur d'arrondi globale de ces mêmes méthodes. Nous prouvons tout d'abord, en section 5.5.1, que nous pouvons déduire une borne d'erreur globale à partir d'une borne sur les erreurs locales relatives de toutes les itérations précédentes. Nous en déduisons ensuite, en section 5.5.2, les bornes d'erreurs globales associées aux méthode d'Euler, de Runge-Kutta 2 et de Runge-Kutta 4.

5.5.1 Passage des erreurs locales aux erreurs globales

Cette section est vouée au lien de dépendance entre erreurs locales et globales. L'analyse proposée inclut la gestion des dépassements de capacité inférieurs lors du passage des erreurs locales aux erreurs globales.

Tout d'abord, nous prouvons un lemme auxiliaire et relativement simple :

Lemme 5.8. *Soit $C \geqslant 0$, $\rho \geqslant 0$. Supposons que $0 < Cu + \rho$. Alors, pour tout $n \in \mathbb{N}$:*

$$(Cu+\rho)^{n+1}\left(\varepsilon_0 + n\frac{Cu|y_0|}{Cu+\rho}\right) + Cu\rho^n|y_0| \leqslant (Cu+\rho)^{n+1}\left(\varepsilon_0 + (n+1)\frac{Cu|y_0|}{Cu+\rho}\right).$$

Démonstration. En simplifiant par $(Cu+\rho)^{n+1}\varepsilon_0$ de chaque côté de l'inégalité, il reste à prouver :

$$(Cu+\rho)^{n+1} n\frac{Cu|y_0|}{Cu+\rho} + Cu\,\rho^n\,|y_0| \leqslant (Cu+\rho)^{n+1}\,(n+1)\frac{Cu|y_0|}{Cu+\rho}.$$

En simplifiant par $(Cu+\rho)^{n+1} n\frac{Cu|y_0|}{Cu+\rho}$, il reste à prouver que :

$$Cu\,\rho^n\,|y_0| \leqslant (Cu+\rho)^{n+1}\frac{Cu|y_0|}{Cu+\rho}.$$

Cela est trivialement vrai si $y_0 = 0$ ou $C = 0$. Dans les autres cas, il reste à prouver :

$$\rho^n \leqslant (Cu+\rho)^{n+1}\frac{1}{Cu+\rho} = (Cu+\rho)^n.$$

Comme $C \geqslant 0$, l'inégalité est vérifiée, ce qui achève la démonstration. □

Le théorème qui suit permet d'obtenir une borne sur l'erreur d'arrondi globale d'une méthode donnée à partir de toutes les erreurs locales précédentes. Dans la section 5.4, la borne sur l'erreur locale à l'itération n dépend de l'occurrence ou non de dépassements de capacité inférieurs. Le caractère stable des méthodes étudiées permet en fait de montrer que, tant que la dernière erreur locale n'est pas impactée par les dépassements de capacité inférieurs, il en est de même pour l'erreur globale.

Théorème 5.9. Passage des erreurs locales aux erreurs globales
Soit $h \in \mathbb{F}$, $\lambda \in \mathbb{R}$. Soit $C \geqslant 0, D \geqslant 0, M > 0$. Considérons une méthode de Runge-Kutta dont la fonction de stabilité est $R(h,\lambda)$. Soit $(\widetilde{y_n})_{n\in\mathbb{N}}$ la suite des solutions calculées par cette méthode. Soient $(\varepsilon_n)_{n\in\mathbb{N}}$ et $(E_n)_{n\in\mathbb{N}}$ les suites des erreurs locales et globales associées à l'implémentation de cette méthode. Supposons que :

- $\forall n \in \mathbb{N}^*, |\widetilde{y_n}| \geqslant M \Rightarrow \varepsilon_n \leqslant Cu|\widetilde{y_{n-1}}|$;
- $\forall n \in \mathbb{N}^*, \varepsilon_n \leqslant Cu|\widetilde{y_{n-1}}| + D\eta$;
- $0 < Cu + |R(h,\lambda)| < 1$.

Alors :

- $\forall n, |\widetilde{y_n}| \geqslant M \Rightarrow |E_n| \leqslant (Cu+|R(h,\lambda)|)^n \left(\varepsilon_0 + n\frac{Cu|y_0|}{Cu+|R(h,\lambda)|}\right)$;
- $\forall n, |E_n| \leqslant (Cu+|R(h,\lambda)|)^n \left(\varepsilon_0 + n\frac{Cu|y_0|}{Cu+|R(h,\lambda)|}\right) + nD\eta$.

Démonstration. Nous procédons par induction sur n. Pour $n = 0$, nous avons :

$$|E_0| = \varepsilon_0 = (Cu+|R(h,\lambda)|)^0\left(\varepsilon_0 + 0\frac{Cu|y_0|}{Cu+|R(h,\lambda)|}\right) + 0D\eta$$

et les deux conclusions du théorème sont trivialement vraies.

Supposons que le résultat est vrai pour un certain $n \in \mathbb{N}$.

- Supposons que $|\widetilde{y_{n+1}}| \geqslant M$.
  Par application du lemme 5.1, nous avons $|\widetilde{y_n}| \geqslant M$. Essayons maintenant de borner $|E_{n+1}|$ par simple dépliage des définitions d'erreurs données par les équations (6.4) et (6.5) et *via* une inégalité triangulaire :

  $$|E_{n+1}| \leqslant |\widetilde{y_{n+1}} - R(h,\lambda)\widetilde{y_n}| + |R(h,\lambda)\widetilde{y_n} - R(h,\lambda)y_n| = \varepsilon_{n+1} + |R(h,\lambda)|\,|E_n|.$$

  Nous bornons ε_{n+1} en utilisant l'hypothèse du théorème :

  $$\varepsilon_{n+1} \leqslant Cu|\widetilde{y_n}| \leqslant Cu\left(|\widetilde{y_n} - y_n| + |y_n|\right) = Cu|E_n| + Cu\,|R(h,\lambda)|^n\,|y_0|. \tag{5.8}$$

  Puis, *via* l'équation (5.8) :

$$|E_{n+1}| \leqslant Cu|E_n| + Cu|R(h,\lambda)|^n|y_0| + |R(h,\lambda)||E_n| = (Cu + |R(h,\lambda)|)|E_n| + Cu|R(h,\lambda)|^n|y_0|.$$

En utilisant l'hypothèse d'induction, nous obtenons :

$$\begin{aligned}|E_{n+1}| &\leqslant (Cu + |R(h,\lambda)|)(Cu + |R(h,\lambda)|)^n \left(\varepsilon_0 + n\,\frac{Cu|y_0|}{Cu + |R(h,\lambda)|}\right) \\ &\quad + Cu\,|R(h,\lambda)|^n\,|y_0| \\ &= (Cu + |R(h,\lambda)|)^{n+1} \left(\varepsilon_0 + n\frac{Cu|y_0|}{Cu + |R(h,\lambda)|}\right) + Cu\,|R(h,\lambda)|^n\,|y_0|.\end{aligned}$$

Puis, en appliquant le lemme 5.8 avec $\rho = |R(h,\lambda)|$:

$$|E_{n+1}| \leqslant (Cu + |R(h,\lambda)|)^{n+1} \left(\varepsilon_0 + (n+1)\frac{Cu|y_0|}{Cu + |R(h,\lambda)|}\right).$$

- Supposons que $|\widetilde{y_{n+1}}|$ est quelconque.
  En bornant ε_{n+1} à partir des hypothèses :

$$\begin{aligned}|E_{n+1}| &\leqslant \varepsilon_{n+1} + |R(h,\lambda)||E_n| \leqslant Cu|\widetilde{y_n}| + D\eta + |R(h,\lambda)||E_n| \\ &\leqslant Cu\,(|\widetilde{y_n} - y_n| + |y_n|) + D\eta + |R(h,\lambda)||E_n| \\ &= Cu|E_n| + Cu\,|R(h,\lambda)|^n|y_0| + D\eta + |R(h,\lambda)||E_n| \\ &= (Cu + |R(h,\lambda)|)|E_n| + Cu|R(h,\lambda)|^n|y_0| + D\eta.\end{aligned}$$

  En utilisant l'hypothèse d'induction, nous obtenons :

$$\begin{aligned}|E_{n+1}| &\leqslant (Cu + |R(h,\lambda)|) \\ &\quad \times \left((Cu + |R(h,\lambda)|)^n \left(\varepsilon_0 + n\tfrac{Cu|y_0|}{Cu+|R(h,\lambda)|}\right) + nD\eta\right) + Cu|R(h,\lambda)|^n|y_0| + D\eta \\ &= (Cu + |R(h,\lambda)|)^{n+1} \left(\varepsilon_0 + n\tfrac{Cu|y_0|}{Cu+|R(h,\lambda)|}\right) \\ &\quad + Cu|R(h,\lambda)|^n\,|y_0| + (Cu + |R(h,\lambda)|)nD\eta + D\eta \\ &\leqslant (Cu + |R(h,\lambda)|)^{n+1} \left(\varepsilon_0 + n\tfrac{Cu|y_0|}{Cu+|R(h,\lambda)|}\right) + Cu|R(h,\lambda)|^n\,|y_0| + (n+1)D\eta.\end{aligned}$$

  En appliquant le lemme 5.8 :

$$|E_{n+1}| \leqslant (Cu + |R(h,\lambda)|)^{n+1} \left(\varepsilon_0 + (n+1)\frac{Cu|y_0|}{Cu + |R(h,\lambda)|}\right) + (n+1)D\eta.$$

□

Ce théorème permet de borner l'erreur d'arrondi globale de façon générique, même en présence de dépassements de capacité inférieurs. Bien que les erreurs locales s'accumulent au cours des itérations, l'erreur globale ne diverge pas de manière trop rapide.

Le phénomène s'explique par la propriété de stabilité linéaire : il y a un effet d'amortissement des premières erreurs dû à la multiplication par $|R(h,\lambda)| < 1$. Ce résultat induit donc une relation intéressante entre la stabilité au sens de l'analyse numérique et la stabilité au sens de l'arithmétique à virgule flottante. En l'absence de dépassement de capacité inférieur, le théorème 5.9 donne également une borne sur l'erreur relative :

$$\left|\frac{\widetilde{y_n} - y_n}{y_n}\right| \leqslant \left(\frac{Cu + |R(h,\lambda)|}{|R(h,\lambda)|}\right)^n \left(\left|\frac{\widetilde{y_0} - y_0}{y_0}\right| + n\frac{Cu}{Cu + |R(h,\lambda)|}\right).$$

Si, de plus, $Cu \ll |R(h,\lambda)|$, l'expression se réduit à :

$$\left|\frac{\widetilde{y_n} - y_n}{y_n}\right| \lesssim \left|\frac{\widetilde{y_0} - y_0}{y_0}\right| + n\frac{Cu}{|R(h,\lambda)|}. \tag{5.9}$$

Les résultats de la section 5.4 montrent qu'en pratique la constante C est de petite taille (au plus 54,47 pour les méthodes de Runge-Kutta considérées). Faire l'hypothèse $Cu \ll |R(h,\lambda)|$ est donc raisonnable et permet d'obtenir une borne d'erreur relative proportionnelle au nombre n d'itérations.

Le théorème 5.9 est un cas particulier du théorème 6.12 qui correspond au cas matriciel et a été formalisé en Coq.

5.5.2 Bornes sur l'erreur globale de méthodes de Runge-Kutta

En section 5.4, nous avons exhibé des bornes sur les erreurs locales de certaines méthodes de Runge-Kutta. Puis, en section 5.5.1, nous avons montré qu'il était possible de borner l'erreur d'arrondi globale à partir des bornes d'erreurs locales des itérations précédentes (théorème 5.9). L'obtention de bornes d'erreurs globales est donc un procédé mécanique et presque immédiat. Dans cette section, nous bornons l'erreur globale induite par les méthodes d'Euler, de Runge-Kutta 2 et de Runge-Kutta 4. Comme dans la section 5.4.2, le format considéré est *binary64*.

Nous commençons par la méthode d'Euler, pour laquelle il faut instancier le théorème 5.9 avec les constantes du lemme 5.5, *i.e.* $C = 9{,}01$, $D = 1{,}01$ et $M = \frac{\xi}{2(1-2u)}$.

Théorème 5.10. Borne sur l'erreur globale de la méthode d'Euler
Soit $h \in \mathbb{F}$, $\lambda \in \mathbb{R}$. *Supposons que* $-2 \leqslant h\lambda \leqslant -2^{-100}$, $9{,}01u + |1 + h\lambda| < 1$ *(stabilité) et* $2^{-60} \leqslant h \leqslant 1$. *Alors, dans le format binary64 :*

- $\forall n, |E_n| \leqslant (9{,}01u + |1 + h\lambda|)^n \left(\varepsilon_0 + n\frac{9{,}01u|y_0|}{9{,}01u+|1+h\lambda|}\right) + 1{,}01n\eta$;
- $\forall n, |\widetilde{y_n}| \geqslant \frac{\xi}{2(1-2u)} \Rightarrow |E_n| \leqslant (9{,}01u + |1 + h\lambda|)^n \left(\varepsilon_0 + n\frac{9{,}01u|y_0|}{9{,}01u+|1+h\lambda|}\right)$.

De même, pour la méthode de Runge-Kutta d'ordre 2, il suffit d'instancier le théorème 5.9 en utilisant les constantes données par le lemme 5.6.

Théorème 5.11. Borne sur l'erreur globale de la méthode RK2
Soit $h \in \mathbb{F}$, $\lambda \in \mathbb{R}$. *Supposons que* $-2 \leqslant h\lambda \leqslant -2^{-100}$, $24{,}03u + \left|1 + h\lambda + \frac{h^2\lambda^2}{2}\right| < 1$ *(stabilité) et* $2^{-60} \leqslant h \leqslant 1$. *Alors, dans le format binary64 :*

- $\forall n, |E_n| \leqslant \left(24{,}03u + \left|1 + h\lambda + \frac{h^2\lambda^2}{2}\right|\right)^n \left(\varepsilon_0 + n\frac{24{,}03u|y_0|}{24{,}03u + \left|1+h\lambda+\frac{h^2\lambda^2}{2}\right|}\right) + 1{,}01n\eta$;
- $\forall n, |\widetilde{y_n}| \geqslant \frac{\xi}{2(1-5{,}01u)} \Rightarrow |E_n| \leqslant \left(24{,}03u + \left|1 + h\lambda + \frac{h^2\lambda^2}{2}\right|\right)^n \left(\varepsilon_0 + n\frac{24{,}03u|y_0|}{24{,}03u + \left|1+h\lambda+\frac{h^2\lambda^2}{2}\right|}\right)$.

Enfin, nous utilisons les constantes exhibées dans le lemme 5.7 pour borner l'erreur globale induite par la méthode RK4. Nous notons $R(h,\lambda) = 1 + h\lambda + \frac{h^2\lambda^2}{2} + \frac{h^3\lambda^3}{12} + \frac{h^4\lambda^4}{24}$.

Théorème 5.12. Borne sur l'erreur globale de la méthode RK4
Soit $h \in \mathbb{F}$, $\lambda \in \mathbb{R}$. *Supposons que* $-2 \leqslant h\lambda \leqslant -2^{-100}$, $54{,}47u + |R(h,\lambda)| < 1$ *(stabilité) et* $2^{-60} \leqslant h \leqslant 1$. *Alors, dans le format binary64 :*

- $\forall n, |E_n| \leqslant (54{,}47u + |R(h,\lambda)|)^n \left(\varepsilon_0 + n\frac{54{,}47u|y_0|}{54{,}47u+|R(h,\lambda)|}\right) + 5{,}34n\eta$;
- $\forall n, |\widetilde{y_n}| \geqslant \frac{3\xi}{1-3{,}01u} \Rightarrow |E_n| \leqslant (54{,}47u + |R(h,\lambda)|)^n \left(\varepsilon_0 + n\frac{54{,}47u|y_0|}{54{,}47u+|R(h,\lambda)|}\right)$.

5.6 Gestion des dépassements de capacité supérieurs

Les méthodes étudiées ne comportant que des additions et multiplications, il est relativement simple de vérifier *a posteriori* l'occurrence d'un dépassement supérieur. Toutefois, comme nous ne considérons que des méthodes stables, le lemme 5.2 assure la décroissance des solutions en l'absence de dépassement de capacité inférieur. Cela permet de prouver l'absence de dépassement supérieur au cours des itérations sur la seule donnée d'une borne raisonnable sur $\widetilde{y_0}$. Ces résultats n'ont pas été généralisés au cas matriciel dans le chapitre 6 et ne sont pour l'heure pas formalisés en Coq.

Nous distinguons les dépassements de capacité locaux et globaux. Nous dirons qu'une méthode provoque un dépassement de capacité local à l'itération n si au moins l'un des calculs intermédiaires de l'itération n provoque un dépassement de capacité supérieur. Pour être plus précis, considérons une méthode de Runge-Kutta caractérisée par la relation $\widetilde{y_{n+1}} = \bigoplus_{i=0}^{s} \widetilde{\alpha_i} \otimes \widetilde{y_n}$, *i.e.*, $\widetilde{y_{n+1}} = (\widetilde{\alpha_0} \otimes \widetilde{y_n} \oplus (\widetilde{\alpha_1} \otimes \widetilde{y_n} \oplus (\cdots \oplus (\widetilde{\alpha_s} \otimes \widetilde{y_n}))\ldots)$. Nous parlerons de dépassement de capacité local si au moins l'une des sous-sommes $\bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes \widetilde{y_n}$ (avec $k \leqslant s$) provoque un dépassement supérieur.

Rappelons que $\Omega = (\beta - \beta^{1-p})\beta^{e_{\max}}$ est le plus grand nombre flottant normalisé.

Lemme 5.13. Dépassement local de capacité supérieur
Soit $s \in \mathbb{N}$. *Soit* $(\widetilde{\alpha_i})_{n\in\mathbb{N}}$ *une suite de flottants. Soit* $y \in \mathbb{F}$. *Soit* $0 < V$. *Supposons que :*

- $\sum_{i=0}^{s} |\widetilde{\alpha_i}| \leqslant V$;
- $|y| \leqslant \frac{\Omega}{(1+(s+1)u)V}$;
- $(s+1)u \leqslant 1$;
- $\xi \leqslant 2^{e_{\max}-p}$.

Alors : $\forall k \in \mathbb{N}, k \leqslant s \Rightarrow \left|\bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y\right| \leqslant \Omega$.

Démonstration. D'après le théorème 2.10 :

$$\left|\bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y - \sum_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y\right| \leqslant \frac{ku}{1+u} \sum_{i=0}^{k} |\widetilde{\alpha_{i+s-k}} \otimes y|.$$

Donc :

$$\left|\bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y\right| \leqslant \frac{ku}{1+u} \sum_{i=0}^{k} |\widetilde{\alpha_{i+s-k}} \otimes y| + \left|\sum_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y\right|.$$

Ainsi, par inégalité triangulaire dans la somme :

$$\left|\bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y\right| \leqslant \left(\frac{ku}{1+u} + 1\right) \sum_{i=0}^{k} |\widetilde{\alpha_{i+s-k}} \otimes y|.$$

Bornons maintenant la somme sus-mentionnée. Nous remarquons qu'un dépassement de capacité inférieur pourrait se produire dans la multiplication par y. C'est la raison pour laquelle un terme $\frac{\eta}{2}$ apparaît pour chaque terme de la sommation :

$$\sum_{i=0}^{k} |\widetilde{\alpha_{i+s-k}} \otimes y| \leqslant \sum_{i=0}^{k} (|\widetilde{\alpha_{i+s-k}} \otimes y - \widetilde{\alpha_{i+s-k}} y| + |\widetilde{\alpha_{i+s-k}} y|) \leqslant \sum_{i=0}^{k} \left(|y||\widetilde{\alpha_{i+s-k}}|(1+u) + \frac{\eta}{2}\right)$$

$$\leqslant \frac{\Omega}{V(1+(s+1)u)} V(1+u) + (k+1)\frac{\eta}{2} = \frac{\Omega}{1+(s+1)u}(1+u) + (k+1)\frac{\eta}{2}.$$

Donc :

$$\left|\bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y\right|$$

$$\leqslant \left(\frac{1+(k+1)u}{1+u}\right)\left(\frac{\Omega}{1+(s+1)u}(1+u) + (k+1)\frac{\eta}{2}\right)$$

$$< \frac{1+(k+1)u}{1+(s+1)u}\Omega + (1+(k+1)u)(k+1)\frac{\eta}{2}.$$

Donc, comme $k \leqslant s$ et $(k+1)u \leqslant (s+1)u \leqslant 1$:

$$\left|\bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y\right| < \Omega + (1+(k+1)u)(k+1)\frac{\eta}{2}$$

$$\leqslant \Omega + \frac{\eta}{u} = \Omega + 2\xi \leqslant \Omega + 2 \times 2^{e_{\max}-p} = \Omega + \mathrm{ulp}(\Omega).$$

Donc, comme $\bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y$ est un nombre flottant, $\left|\bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes y\right| \leqslant \Omega$. □

Nous en déduisons le résultat suivant :

Théorème 5.14. Dépassement global de capacité supérieur
Soit $C \geqslant 0$, $M > 0$. *Considérons une méthode de Runge-Kutta définie par la relation* $\forall n \in \mathbb{N}, \widetilde{y_{n+1}} = \bigoplus_{i=0}^{s} \widetilde{\alpha_i} \otimes \widetilde{y_n}$. *Supposons que :*

- $\sum_{i=0}^{s} |\widetilde{\alpha_i}| \leqslant V$;
- $(s+1)u \leqslant 1$;
- $\forall n \in \mathbb{N}^*, |\widetilde{y_n}| \geqslant M \Rightarrow \varepsilon_n \leqslant Cu|\widetilde{y_{n-1}}|$;
- $Cu + |R(h,\lambda)| \leqslant 1$;
- $M \leqslant \frac{\Omega}{(1+(s+1)u)V}$;
- $\xi \leqslant 2^{e_{\max}-p}$;
- $|\widetilde{y_0}| \leqslant \min\left(\Omega, \frac{\Omega}{(1+(s+1)u)V}\right)$.

Alors, pour tout n, *il n'y a aucun dépassement de capacité supérieur en calculant* $\widetilde{y_n}$.

Démonstration. La démonstration se fait par induction sur n.

Si $n = 0$, comme $\widetilde{y_0}$ est fini, il n'y a pas de dépassement de capacité supérieur.

Soit n un entier donné. Supposons qu'aucun dépassement de capacité supérieur ne s'est produit après n itérations (hypothèse d'induction). Si $|\widetilde{y_n}| \geqslant M$, alors d'après le lemme 5.2, $|\widetilde{y_n}| \leqslant |\widetilde{y_0}| \leqslant \frac{\Omega}{(1+(s+1)u)V}$. Si $|\widetilde{y_n}| < M$ alors nous avons également $|\widetilde{y_n}| < \frac{\Omega}{(1+(s+1)u)V}$. Donc, dans les deux cas, d'après le lemme 5.13, pour tout $k \leqslant s$:

$$\left| \bigoplus_{i=0}^{k} \widetilde{\alpha_{i+s-k}} \otimes \widetilde{y_n} \right| \leqslant \Omega.$$

Donc, il n'y a pas de dépassement de capacité supérieur dans le calcul de $\widetilde{y_{n+1}}$. Combiné à l'hypothèse d'induction, ce résultat achève la démonstration. □

Montrons que les hypothèses du Théoreme 5.14 sont sensées :

- Pour prévenir le risque de dépassement de capacité supérieur, il suffit de vérifier que $|\widetilde{y_0}|$ n'est pas trop grand. Le tableau 5.4 donne les valeurs de s et un ordre de grandeur de V et $\frac{\Omega}{(1+(s+1)u)V}$, en choisissant V égal à sa valeur minimale $\sum_{i=0}^{s} |\widetilde{\alpha_i}|$ (calculée *via* les contraintes sur $h\lambda$), pour des méthodes classiques implémentées au format *binary64*. En fait, les problèmes résolus par des méthodes de Runge-Kutta font généralement apparaître des constantes de taille raisonnable. Par exemple, même en astrophysique, les conditions initiales n'excèdent pas 10^{24} [21], qui est une valeur bien inférieure par rapport aux bornes du tableau 5.4.
- De plus, le théorème 5.14 utilise l'hypothèse $\xi \leqslant 2^{e_{\max}-p}$. En fait, tous les formats utiles d'arithmétique à virgule flottante vérifient cette hypothèse. Par exemple, dans le format *binary64*, $\xi = 2^{-1022}$ et $2^{e_{\max}-p} = 2^{970}$.

5.7 Résultats empiriques

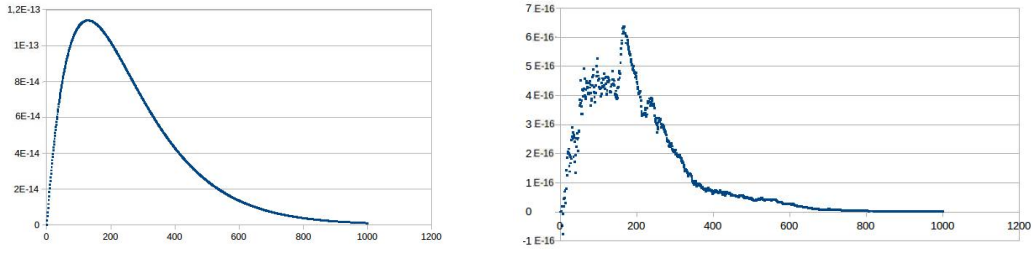
La méthodologie adoptée étant relativement générique, les bornes fournies peuvent s'avérer pessimistes. En effet, nous ne tenons pas compte des particularités de chaque équation différentielle et de chaque méthode numérique. Les bornes que nos résultats permettent d'obtenir sont valables « dans le pire cas ».

Tableau 5.4 – Valeurs de V, s, et $\frac{\Omega}{(1+(s+1)u)V}$ pour des méthodes classiques en *binary*64

Method	V	s	$\frac{\Omega}{(1+(s+1)u)V}$
Euler	3	2	$\frac{\Omega}{3+9u} \simeq 5{,}99 \times 10^{307}$
RK2	5	3	$\frac{\Omega}{5+20u} \simeq 3{,}6 \times 10^{307}$
RK4	16,5	11	$\frac{\Omega}{16{,}5+198u} \simeq 1{,}09 \times 10^{307}$

Nous avons néanmoins souhaité comparer ces bornes aux erreurs d'arrondi effectivement commises par l'implémentation de méthodes de Runge-Kutta sur des systèmes dynamiques particuliers. Il faut pour cela pouvoir se baser sur une implémentation « idéale », *i.e.* en précision infinie, des algorithmes décrits en section 5.1.2. Plutôt que d'implémenter les méthodes à l'aide de nombres réels constructifs, nous avons jugé suffisant d'utiliser MPFR avec une précision de 1000 bits. Nous comparons les solutions ainsi obtenues aux résultats d'une implémentation en double précision. Prenons l'exemple de l'équation $\dot{y} = -\frac{y}{2}$, *i.e.* $\lambda = -0{,}5$ avec la condition initiale $y_0 = 1$. Nous résolvons le système avec la méthode de Runge-Kutta d'ordre 2 en choisissant un pas $h = \frac{1}{64}$ sur $n = 1000$ itérations. La figure 5.2(a) montre le résultat de l'évaluation de la borne obtenue (théorème 5.11) sur l'exemple choisi. Pour n petit, une bosse apparaît, puis la borne d'erreur décroît rapidement. La borne calculée étant égale à $(Cu + |R(h,\lambda)|)^n \, n \frac{Cu|y_0|}{Cu+|R(h,\lambda)|}$ avec $C \approx 24$, le terme dominant est de la forme $nCu|y_0|$ pour n petit. En revanche, quand n devient grand, le terme $(Cu + |R(h,\lambda)|)^n$ domine et la borne d'erreur décroît (quasi-) exponentiellement. Nous comparons cette borne à l'erreur signée effective (figure 5.2(b)).

Les axes des ordonnées des figures 5.2(a) et 5.2(b) ne sont pas à la même échelle. Il apparaît que l'erreur est petite par rapport à la borne obtenue, avec un rapport d'environ 180. Cela s'explique par le fait que nous fournissons une analyse « pire cas », pour tout h, pour tout λ et sans prendre en compte d'éventuelles compensations d'erreurs au cours des itérations. Notons néanmoins que l'erreur effective et la borne d'erreur semblent évoluer de façon similaire, avec une forte hausse puis une décroissance rapide.



(a) Borne d'erreur d'arrondi (b) Erreurs d'arrondi signées

FIGURE 5.2 – Borne d'erreur d'arrondi et erreur signée pour RK2 appliquée au système $\dot{y} = -\frac{y}{2}$, *i.e.* $\lambda = -0{,}5$ avec $y_0 = 1$ et $h = \frac{1}{64}$ sur $n = 1000$ itérations.

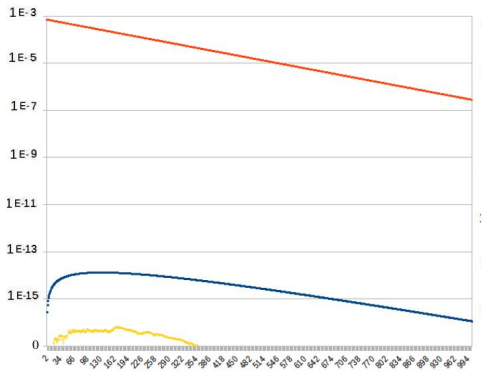


FIGURE 5.3 – Exemple de RK2 : erreur de méthode (en rouge), borne d'erreur d'arrondi (en bleu) et erreur signée (en jaune) sur une échelle logarithmique

Nous comparons également les erreurs d'arrondi aux erreurs de méthode des schémas de Runge-Kutta. La figure 5.3 présente, sur une échelle logarithmique, l'erreur de méthode pour la méthode RK2 (obtenue en comparant les implémentations MPFR sur 1000 bits de la méthode et de la fonction exponentielle) ainsi que la borne d'erreur d'arrondi et l'erreur d'arrondi obtenue en pratique. Bien que pessimistes, les bornes exhibées restent négligeables par rapport à l'erreur de méthode. Ces bornes suffisent donc à garantir que les erreurs d'arrondi ne faussent pas significativement les résultats.

5.8 Conclusion

Nous avons proposé une analyse d'erreurs d'arrondi applicable à une classe importante de méthodes d'intégration numérique des équations différentielles dans le cas linéaire unidimensionnel. Cette analyse d'erreurs à grains fins permet d'obtenir des bornes fines et tenant compte d'éventuels dépassements de capacité inférieurs. Par ailleurs, nous avons obtenu une condition suffisante sur $|y_0|$ permettant de se prémunir de tout risque de dépassement de capacité supérieur au cours des itérations d'une méthode.

La méthodologie adoptée est relativement mécanique et repose sur des résultats génériques. Nous avons de plus déroulé l'analyse d'erreurs sur quelques exemples de méthodes numériques qui sont très fréquemment utilisées en pratique, comme la méthode d'Euler, la méthode du point-milieu (RK2) et la méthode RK4.

En pratique, l'utilisation de la fonction exponentielle est souvent préférée à la résolution numérique d'équations différentielles linéaires dans le cas unidimensionnel. Le rôle principal de ce chapitre est en fait d'introduire une méthodologie ainsi que certaines définitions (erreurs locales, erreurs globales, etc) comme premier pas vers l'analyse d'erreurs dans le cas des systèmes linéaires matriciels, dont la résolution numérique par les méthodes de Runge-Kutta est plus courante et qui fait l'objet du chapitre 6.

Chapitre 6

Erreurs d'arrondi des méthodes de Runge-Kutta : cas matriciel et formalisation

Le chapitre 5 a permis d'introduire une méthodologie générique permettant de borner les erreurs d'arrondi de méthodes de Runge-Kutta explicites appliquées à des systèmes linéaires scalaires. Nous étendons cette méthodologie aux systèmes linéaires matriciels de la forme $y' = Ay$ (avec A une matrice carrée). Il existe en effet de nombreux cas d'utilisation de ce type d'équations pour modéliser des problèmes physiques, *e.g.* en science du nucléaire [12] ou en mécanique quantique [172]. Nous proposons de plus une formalisation en Coq de l'ensemble des résultats dans le cas matriciel, ainsi que de leur application à des méthodes de Runge-Kutta classiques. Nous utilisons Flocq [37, 38] et les matrices de la bibliothèque Mathematical Components [144].

La première étape importante est la formalisation d'éléments d'analyse matricielle en Coq. La bibliothèque Mathematical Components [144] est riche d'une bibliothèque de vecteurs et matrices comportant un nombre important de résultats (voir section 2.2.3). Il faut néanmoins que nous formalisions l'implémentation des opérations matricielles en arithmétique à virgule flottante ainsi que des bornes sur les erreurs d'arrondi associées. Cette étape de la formalisation est la plus technique car elle repose sur des manipulations fines de matrices, de vecteurs et de normes qui leur sont associées.

La seconde étape consiste à vérifier formellement les résultats génériques permettant de construire les erreurs locales et globales associées aux implémentations de méthodes de Runge-Kutta. Les démonstrations de ces résultats suivent des raisonnements relativement naturels et présentent assez peu de difficultés techniques. Elles peuvent cependant s'avérer fastidieuses et calculatoires, ce qui peut conduire à des erreurs de raisonnement que l'usage de Coq permet de mieux contrôler.

Enfin, la dernière étape de la formalisation consiste à instancier les résultats aux méthodes d'Euler et de Runge-Kutta 2. Ces applications sont calculatoires et très difficiles à mener à la main. L'assistant de preuves Coq s'impose là encore comme un allié.

Nous utilisons des tactiques, *e.g.* `eapply` et `interval_tactic` [150], qui permettent d'automatiser une partie des raisonnements et de simplifier les bornes obtenues.

Ce chapitre est organisé de la manière suivante. La section 6.1 pose les hypothèses sur le format d'arithmétique que nous considérons et rappelle ou introduit les notations utilisées dans la suite du chapitre. La section 6.2 définit les méthodes que nous étudions ainsi que leur implémentation dans le cadre de la résolution des systèmes matriciels. Nous décrivons ensuite, en section 6.3, les choix menés pour l'analyse d'erreurs ainsi que les différentes étapes de la méthodologie. La section 6.4 est vouée à la formalisation en Coq des méthodes numériques et de leur implémentation, ainsi qu'à la spécification des concepts d'erreurs locales et globales. La section 6.5 présente la formalisation de la norme infinie de Hölder, utilisée pour borner les erreurs d'arrondi. La section 6.6 décrit quant à elle la formalisation de résultats bornant les erreurs d'arrondi associées aux produits de matrices. La section 6.7 présente les résultats génériques permettant de borner les erreurs locales de méthodes de Runge-Kutta, ainsi que l'application de ces résultats à des méthodes classiques. La section 6.8 est quant à elle vouée aux bornes sur les erreurs globales, *i.e.* accumulées au cours des itérations.

6.1 Hypothèses et notations

Dans l'ensemble du chapitre, le format flottant considéré est le format `FLT` de Flocq (voir [37, 38] et section 2.2.4). La base et la précision sont des paramètres de l'analyse, nous tenons compte d'éventuels dépassements graduels inférieurs mais ne tenons pas compte des dépassements supérieurs. Le mode d'arrondi utilisé est l'arrondi au plus proche, la règle de bris d'égalité étant quelconque. Le format flottant utilisé dans ce chapitre sera désigné par la notation mathématique $\mathbb{F}$. Les notations $u = \frac{1}{2}\beta^{1-p}$ et $\eta = \beta^{e_{\min}-p+1}$ correspondent respectivement à l'unité d'arrondi et au plus petit nombre flottant dénormalisé (voir section 2.1 pour une définition précise) associés à $\mathbb{F}$.

Nous utiliserons les notations Coq `(+)` et `(x)` pour désigner l'arrondi au plus proche de l'addition et de la multiplication dans le format $\mathbb{F}$, *i.e.* les opérations $\oplus$ et $\otimes$. Le prédicat `format : R -> Prop` caractérise les nombres réels représentables dans $\mathbb{F}$.

La notation $\oplus$ sera surchargée pour faire référence à la somme arrondie de deux vecteurs. De même, la notation $\otimes$ (notée `(x)_m` en Coq) est utilisée pour le produit de matrices, avec par convention un ordre d'évaluation de la droite vers la gauche pour les sommations que comporte le produit matriciel. La notation $\otimes$ sera également utilisée pour faire référence à la multiplication d'un scalaire et d'une matrice, *i.e.* pour tout réel a et toute matrice A, pour tout i, j, $(a \otimes A)_{ij} = a \otimes A_{ij}$.

Pour rappel, la notation $\circ[\cdots]$ signifie que toutes les opérations à l'intérieur des crochets sont arrondies, avec par défaut un parenthésage de la droite vers la gauche.

Comme dans le chapitre 5, nous supposerons que le pas d'intégration h est exactement représentable dans $\mathbb{F}$.

6.2 Systèmes et méthodes étudiés

6.2.1 Résolution numérique des systèmes linéaires matriciels par des méthodes de Runge-Kutta

Nous nous intéressons à la résolution numérique de systèmes linéaires multidimensionnels, aussi appelés systèmes matriciels :

$$\dot{y} = Ay \tag{6.1}$$

avec $y \in \mathbb{R}^d$ et $A \in \mathbb{R}^{d\times d}$ une matrice carrée de taille d.

L'application d'une méthode de Runge-Kutta explicite sur le problème matriciel de l'équation (6.1) définit la relation :

$$y_{n+1} = R(h, A)y_n. \tag{6.2}$$

Par exemple, dans le cas de la méthode d'Euler, la relation de récurrence est :

Exemple 6.1. *Application de la méthode d'Euler explicite sur l'équation* (6.1)

$$y_{n+1} = y_n + hAy_n = (I + hA)y_n = R_{Euler}(h, A)y_n.$$

De même, lorsque la méthode de Runge-Kutta d'ordre 2 est appliquée à l'équation (6.1), la relation obtenue est :

Exemple 6.2. *Application de la méthode RK2 explicite sur l'équation* (6.1)

$$k_1 = Ay_n \qquad k_2 = A(y_n + \frac{1}{2}hk_1) = (A + \frac{1}{2}hA^2)y_n$$

$$y_{n+1} = y_n + hk_2 = (1 + hA + \frac{1}{2}h^2A^2)y_n = R_{RK2}(h, A)y_n. \tag{6.3}$$

6.2.2 Implémentations des méthodes de Runge-Kutta

Comme pour le cas unidimensionnel (section 5.1.2), une méthode de Runge-Kutta peut être implémentée de diverses manières. Pour un système linéaire matriciel $y' = Ay$ et un pas d'intégration h, une implémentation est caractérisée par une relation $\widetilde{y_{n+1}} = \widetilde{R}(h, \widetilde{A}, \widetilde{y_n})$ qui permet d'obtenir la solution calculée $\widetilde{y_{n+1}}$ en fonction du pas d'intégration h, d'une approximation $\widetilde{A}$ de la matrice A et de la solution $\widetilde{y_n}$ calculée à l'itération précédente.

Nous nous intéressons à une implémentation particulière des méthodes de Runge-Kutta, que nous qualifions d'implémentation « à la Horner » par analogie avec l'évaluation à la Horner des polynômes. Les implémentations « à la Horner » des méthodes d'Euler et de Runge-Kutta 2 sont définies ci-dessous.

Exemple 6.3. *L'application de la méthode d'Euler sur l'équation différentielle avec condition initiale définie par l'équation* (6.1) *conduit à l'implémentation « à la Horner » :*

$$\widetilde{y_{n+1}} = \widetilde{R_{\text{Euler}}}(h, \widetilde{A}, \widetilde{y_n}) = \circ\left[\widetilde{y_n} + (h\widetilde{A})\widetilde{y_n}\right].$$

Exemple 6.4. *L'application de la méthode RK2 sur l'équation différentielle avec condition initiale définie par l'équation* (6.1) *conduit à l'implémentation « à la Horner » :*

$$\widetilde{y_{n+1}} = \widetilde{R_{\text{RK2}}}(h, \widetilde{A}, \widetilde{y_n}) = \circ\left[\widetilde{y_n} + (h\widetilde{A})\left(\widetilde{y_n} + \left(\frac{h}{2}\widetilde{A}\right)\widetilde{y_n}\right)\right].$$

Ces implémentations sont obtenues en sommant l'un après l'autre les termes k_i utilisés dans la construction des méthodes de Runge-Kutta. En ce sens, elles sont proches des implémentations que proposent naïvement certains solveurs d'équations différentielles, *e.g.* ceux qu'embarque l'outil Simulink.

Dans ce chapitre, nous proposons une analyse d'erreurs générique pour les implémentations de méthodes de Runge-Kutta mais nous intéressons plus particulièrement aux évaluations « à la Horner » décrites ci-dessus.

6.3 Approche choisie et méthodologie

Dans cette section, nous présentons les différentes étapes de la méthodologie développée pour l'analyse des erreurs d'arrondi des méthodes de Runge-Kutta appliquées aux systèmes multidimensionnels.

6.3.1 Choix de l'approche d'analyse et de la norme

Analyse par normes ou par composantes

Nous avons tout d'abord été confronté au choix d'une approche pour l'analyse d'erreurs. Dans le cas des systèmes multidimensionnels, les solutions calculées sont des vecteurs $y_n \in \mathbb{R}^d$. Lorsque ces vecteurs sont calculés, ils sont entachés d'erreurs d'arrondi et nous obtenons un vecteur approché $\widetilde{y_n}$ qui peut être vu comme la somme de y_n et d'un *vecteur d'erreur* e_n, *i.e.* $\widetilde{y_n} = y_n + e_n$.

Nous avons identifié deux approches classiques permettant de mesurer le vecteur d'erreur. La première approche constitue ce que nous appelons l'analyse *par normes*. Elle consiste à considérer les vecteurs et matrices comme des points des espaces respectifs $\mathbb{R}^d$ et $\mathbb{R}^{d\times d}$ puis à mesurer l'amplitude des vecteurs d'erreurs en leur appliquant une norme bien choisie. La seconde approche est appelée analyse *par composantes* et consiste à observer la répartition des erreurs sur les différentes composantes du vecteur d'erreur.

Dans un article de 1994 [106], Higham donne une intuition de l'analyse par normes :

> *Matrix analysis would not have developed into the vast subject it is today without the concept of representing a matrix by a single symbol. Similarly, perturbation theory would not be such a rich and useful area if it were not for norms. A norm compresses the mn numbers in an m × n matrix into a single scalar measure of size, enabling a perturbation result to be presented in a form that is easy to interpret and gives insight.*
>
> — Nicholas J. Higham

Higham expose néanmoins quelques faiblesses de l'analyse par normes [106], qui ne permet pas de connaître la manière dont sont distribuées les erreurs sur les composantes du vecteur d'erreur et qui mène à des bornes moins fines, ce qui s'explique par la nécessité d'appliquer de nombreuses inégalités, notamment triangulaires. L'analyse par composantes permet de pallier ces faiblesses et est fréquemment utilisée depuis le début des années 1990 comme l'atteste la revue de la littérature proposée par Higham [106].

Nous avons privilégié l'usage d'une analyse par normes car les systèmes auxquels nous nous intéressons sont quelconques (*e.g.* les matrices ne sont pas supposées creuses) et que nous proposons une analyse *dans le pire cas*. Ce choix permet de réutiliser en grande partie la méthodologie adoptée dans le chapitre 5, la norme ayant un rôle proche de celui de la valeur absolue dans le cas scalaire. Nous pensons néanmoins qu'une analyse par composantes aurait pu être menée, au prix de raisonnements plus fastidieux.

Choix d'une norme adaptée

L'adoption d'une analyse par normes suppose le choix d'une norme adaptée. Les normes de Hölder font partie des normes les plus utilisées. Nous pouvons par exemple citer la norme euclidienne $\|\cdot\|_2$ définie par $\|v\|_2 = \sqrt{\sum_{i<d} v_i^2}$. Si cette norme est intuitive géométriquement, elle est difficile à manipuler et peu utilisée en analyse d'erreurs.

Nous avons identifié deux normes plus adaptées à l'analyse d'erreurs, à savoir les normes $\|\cdot\|_1$ et $\|\cdot\|_\infty$ de Hölder, respectivement définies par :

$$\forall v \in \mathbb{R}^d, \|v\|_1 = \sum_{i<d} |v_i|, \qquad \|v\|_\infty = \max_{i<d} |v_i|.$$

Ces deux normes donnent une assez bonne idée de l'amplitude des erreurs commises et permettent de majorer les erreurs commises sur n'importe quelle composante du vecteur d'erreur. Par ailleurs, leur expression est relativement simple. En outre, les normes matricielles subordonnées à ces normes vectorielles (voir section 6.5.2) s'expriment facilement en fonction des coefficients de matrices. Nous faisons le choix de travailler avec la norme $\|\cdot\|_\infty$ car celle-ci exprime la magnitude de l'erreur sur la « pire composante »[1].

1. Nous aurions pu mener une analyse similaire avec la norme $\|\cdot\|_1$, dont les propriétés sont proches et dont la norme subordonnée (voir section 6.5.2) $|||\cdot|||_1$ vérifie pour toute matrice A, $|||A|||_1 = |||A^\intercal|||_\infty$.

6.3.2 Méthodologie pour borner les erreurs d'arrondi

Erreurs locales et globales

Comme dans le cas scalaire, nous distinguons les erreurs d'arrondi dites locales et globales. Nous souhaitons borner l'erreur d'arrondi accumulée après n itérations d'une méthode numérique implémentée en arithmétique à virgule flottante. Il s'agit de l'erreur d'arrondi globale (en norme), notée E_n et définie comme :

$$E_n = \|\widetilde{y_n} - y_n\|_\infty = \|\widetilde{R}(h, \widetilde{A}, \widetilde{y_n}) - R(h, A)y_n\|_\infty. \tag{6.4}$$

L'erreur locale à l'itération n correspond à l'erreur d'arrondi commise au cours de cette seule itération de la méthode. Elle est notée ε_n et est définie comme :

$$\varepsilon_0 = \|\widetilde{y_0} - y_0\|_\infty \qquad \forall n \in \mathbb{N}^*, \varepsilon_n = \|\widetilde{R}(h, \widetilde{A}, \widetilde{y_{n-1}}) - R(h, A)\widetilde{y_{n-1}}\|_\infty. \tag{6.5}$$

Étapes de l'analyse d'erreurs

La méthodologie utilisée pour borner l'erreur globale d'une méthode numérique est assez similaire à celle adoptée dans le cas unidimensionnel (section 5.2), à quelques adaptations près :

- nous calculons une borne sur l'erreur d'arrondi $E_0 = \varepsilon_0$ commise lors du calcul de la valeur initiale y_0 ;
- nous construisons une borne sur les erreurs locales ε_n en appliquant mécaniquement plusieurs fois un lemme technique, qui permet de gérer les dépassements graduels inférieurs. Ce lemme est générique et couvre diverses formes d'implémentations, dont les implémentations « à la Horner ». Nous obtenons un résultat de la forme $\forall n, \varepsilon_n \leqslant Cu\|\widetilde{y_{n-1}}\|_\infty + D\eta$ avec C et D des constantes ;
- comme dans le cas unidimensionnel, nous utilisons un théorème générique (théorème 6.12) qui permet de construire une borne sur l'erreur globale à partir des bornes sur les erreurs locales précédentes, en tenant compte d'éventuels dépassements de capacité inférieurs. Nous instancions ce théorème avec les constantes C et D obtenues précédemment.

6.4 Formalisation des méthodes et définitions des erreurs

6.4.1 Formalisation des méthodes et implémentations

Pour une dimension fixée, nous définissons un type `Sc` (pour Schéma) qui caractérise les implémentations de méthodes de Runge-Kutta par la relation de récurrence liant

deux itérations consécutives y_n et y_{n+1}. Dans le code Coq, la notation `d.+1` permet de ne gérer que les vecteurs de taille non nulle.

```
Definition Sc (d : nat) : Type := (R → R) → cV_d.+1 → cV_d.+1.
```

Un élément `mth : Sc d` prend pour paramètres :

- une fonction $\mathscr{W} \in \mathbb{R} \to \mathbb{R}$ que nous appelerons *fonction d'arrondi* par abus de langage. Il peut en effet s'agir soit d'un arrondi soit de la fonction identité (qui permet de caractériser les calculs exacts, sans arrondi) ;
- la solution $\widetilde{y_n} \in \mathbb{R}^d$ calculée à l'itération précédente.

Ainsi, `mth d W` permet d'obtenir $\widetilde{y_{n+1}}$ à partir de $\widetilde{y_n}$.

À titre d'exemples, nous avons défini les implémentations « à la Horner » des méthodes d'Euler et RK2 pour un système de dimension d. Nous introduisons tout d'abord quelques définitions et notations pour les opérations matricielles « arrondies » par la fonction $\mathscr{W}$, *i.e.* le résultat de chaque opération élémentaire est arrondi par $\mathscr{W}$:

```
Definition W_plus {n m}
    (W : R → R) (A : 'M[R]_(m,n)) (B : 'M[R]_(m,n)) : 'M[R]_(m,n)
  := \matrix_(i,j) (W (A i j + B i j)).
Notation "A (+)_[ W ] B" := (W_plus W A B) (at level 40).

Definition W_mult {n m p}
    (W : R → R) (A : 'M[R]_(m, n)) (B : 'M[R]_(n, p)) : 'M[R]_(m, p)
  := \matrix_(i,j) \big[(fun x y ⇒ W (x + y))/0]_k (W (A i k * B k j)).
Notation "A (x)_[ W ] B" := (W_mult W A B) (at level 45).
```

Le mot-clef `level` permet de spécifier le niveau de priorité d'une notation (plus la valeur est petite, plus le niveau de priorité est élevé).

Les matrices `A (+)_[ W ] B` et `A (x)_[ W ] B` désignent respectivement la somme et le produit de `A` et `B`, en considérant que toutes les opérations élémentaires intervenant dans les calculs intermédiaires ont été « arrondies » par la fonction $\mathscr{W}$. Ainsi, en notant `id` la fonction identité, `A (x)_[ id ] B` est le produit matriciel exact. De même, `A (x)_[ rnd ] B` est le produit matriciel suivant une fonction d'arrondi `rnd` donnée. Par ailleurs, `map_mx W A` désigne l'application de `W` à l'ensemble des coefficients de `A`.

Les implémentations à la Horner des méthodes d'Euler et RK2 pour un système de dimension d sont respectivement définies comme :

```
Definition Euler {d} (h : R) (A : 'M_d.+1) : Sc d := fun W : R → R ⇒
  let A_2 := map_mx W (h * (map_mx W A)) in          (* h ⊗ Ã *)
      (fun y ⇒ y (+)_[ W ] (A_2 (x)_[ W ] y)).

Definition RK2 {d} (h : R) (A : 'M_d.+1) : Sc d := fun W : R → R ⇒
  let A_21 := map_mx W (h * (map_mx W A)) in          (* h ⊗ Ã *)
  let A_22 := map_mx W ((W (h/2)) * (map_mx W A)) in  (* h/2 ⊗ Ã *)
    (fun y ⇒ y (+)_[ W ] (A_21 (x)_[ W ]
                          (y (+)_[ W ] (A_22 (x)_[ W ] y)))).
```

Ainsi, `Euler h A id` correspond à un pas d'itération de la méthode d'Euler appliquée au système $y' = Ay$ avec un pas d'intégration h, implémentée en précision infinie. De même, pour le format `FLT` de Flocq, `Euler h A rnd` correspond à l'implémentation « à la Horner » en arithmétique à virgule flottante avec la fonction d'arrondi `rnd`.

Nous définissons ensuite l'évaluation de la méthode numérique `mth : Sc d` sur n itérations à partir d'une condition initiale `y0` :

```
Definition meth_iter mth n y0 W := iter n (mth d W) (map_mx W y0).
```

Notons que l'itérateur a pour élément de base le vecteur `map_mx W y0`, *i.e.* le vecteur `y0` dont toutes les composantes ont été arrondies par la fonction `W`.

6.4.2 Définition des erreurs d'arrondi

Ensuite, nous définissons formellement les erreurs d'arrondi associées aux méthodes de Runge-Kutta définies en section 6.4.1. Comme nous adoptons la méthodologie proposée en section 6.3, nous distinguons les erreurs locales et globales.

L'erreur locale (à l'itération $n+1$) associée à une méthode de Runge-Kutta caractérisée par `mth : Sc d` avec la condition initiale `y0` est définie comme :

```
Definition error_loc (mth : Sc d) n (y0 : cV_d.+1) (W : R → R)
  := vec_norm (mth W (meth_iter mth n y0 W) -
               mth (fun x ⇒ x) (meth_iter mth n y0 W)).
    (*ε_{n+1} = ‖R̃(h, Ã, ỹ_n) − R(h, A)ỹ_n‖_∞*)
```

Nous définissons ensuite l'erreur globale après n itérations de la méthode :

```
Definition error_glob (mth : Sc d) n (y0 : cV_d.+1) (W : R → R)
  := vec_norm (meth_iter mth n y0 W - meth_iter mth n y0 (fun x ⇒ x)).
    (*E_n = ‖ỹ_n − y_n‖_∞*)
```

Pour l'erreur locale comme pour l'erreur globale, l'idée générale est d'utiliser le caractère générique de la *fonction d'arrondi* $\mathscr{W}$ pour comparer les implémentations en précision infinie (*i.e.* $\mathscr{W}$ est la fonction identité) et avec arrondi (*i.e.* $\mathscr{W}$ est un arrondi dans le format flottant considéré).

6.5 Normes vectorielles et matricielles en Coq

Comme évoqué en section 6.3, nous souhaitons mener une analyse d'erreurs par normes, la norme choisie étant la norme infinie de Hölder, notée $\|\cdot\|_\infty$. Dans cette section, nous décrivons la formalisation de cette norme et de la norme matricielle qui lui est subordonnée. Nous présentons également la démonstration formelle des propriétés fondamentales de ces normes, notamment la propriété de sous-multiplicativité (voir section 6.5.3).

6.5.1 Norme vectorielle infinie

Pour formaliser la norme $\|\cdot\|_\infty$, nous utilisons un grand opérateur (voir section 2.2.3) itérant sur l'opération binaire `Rmax` caractérisant le maximum de deux nombres réels.

```
Definition vec_norm {d} : 'cV[R]_d → R :=
      fun v ⇒ \big[Rmax/0]_(i < d) (Rabs (v i)).
```

L'utilisation de la valeur 0 comme valeur par défaut associée au grand opérateur itérant l'opération `Rmax` peut *a priori* surprendre. Cependant, dans ce cadre, les éléments sur lesquels itère le grand opérateur sont les valeurs absolues des composantes d'un vecteur, qui sont toujours positives.

Nous vérifions sans difficulté particulière que `vec_norm` satisfait bien les propriétés mathématiques d'une norme, *i.e.* :

- la norme du vecteur dont toutes les composantes sont nulles est égale à 0 :

  ```
  Lemma vec_norm0 {d} : vec_norm (vec0 d) = 0.
  ```

- la norme satisfait la propriété d'homogénéité :

  ```
  Lemma vec_norm_scal {d} : forall (v : 'cV[R]_d) (k:R),
       vec_norm (map_mx (fun x ⇒ k*x) v) = Rabs k * vec_norm v.
  ```

- la norme satisfait la propriété d'inégalité triangulaire :

  ```
  Lemma vec_norm_triang {d} : forall (a b : 'cV[R]_d),
       vec_norm (a + b) <= vec_norm a + vec_norm b.
  ```

6.5.2 Norme matricielle subordonnée

À partir d'une norme vectorielle $\|\cdot\|$ sur $\mathbb{R}^d$, il est possible de définir une norme $|||\cdot|||$ sur $\mathbb{R}^{d\times d}$ dite subordonnée à $\|\cdot\|$ et définie comme :

$$\forall A \in \mathbb{R}^{d\times d}, |||A||| = \sup_{v\in\mathbb{R}^d\setminus\{0\}} \frac{\|Av\|}{\|v\|}.$$

Dans le cas de la norme $\|\cdot\|_\infty$, la norme subordonnée $|||\cdot|||_\infty$ peut être obtenue *via* une caractérisation directe à partir des coefficients de la matrice :

$$\forall A \in \mathbb{R}^{d\times d}, |||A|||_\infty = \max_{i<d} \sum_{j<d} |A_{ij}|.$$

Nous avons choisi de ne pas définir le concept générique de norme subordonnée et utilisons la caractérisation directe de $|||\cdot|||_\infty$. La formalisation associée à cette norme repose sur l'usage de grands opérateurs imbriqués, utilisés pour itérer les opérations de sommation et de maximum :

```
Definition matrix_norm {d} : 'M[R]_d → R :=
      fun A ⇒ \big[Rmax/0]_(i < d)
                  (\big[Rplus/0]_(j < d) (Rabs (A i j))).
```

De la même façon que pour la norme vectorielle $\|\cdot\|_\infty$, nous démontrons que $|||\cdot|||_\infty$ est bien une norme pour l'ensemble des matrices carrées de taille d.

6.5.3 Propriétés de sous-multiplicativité

Une propriété essentielle des normes matricielles qui permet de mener à bien une analyse d'erreurs est la sous-multiplicativité. Cette propriété est vérifiée par la norme subordonnée à la norme infinie pour les produits matrice-vecteur et matrice-matrice.

Commençons par énoncer la sous-multiplicativité vis-à-vis du produit d'une matrice et d'un vecteur, dont la démonstration formelle repose essentiellement sur l'application des propriétés fondamentales des normes :

Lemme 6.1. Sous-multiplicativité de $|||\cdot|||_\infty$, produit matrice-vecteur 
Soient $d \in \mathbb{N}^*$, $v \in \mathbb{R}^d$ et $A \in \mathbb{R}^{d\times d}$. Alors :

$$\|A \times v\|_\infty \leqslant |||A|||_\infty \|v\|_\infty.$$

Démonstration. Par définition de $\|\cdot\|_\infty$ et du produit matrice-vecteur :

$$\|A \times v\|_\infty = \max_{i<d} |(A \times v)_i| = \max_{i<d} \left| \sum_{k<d} A_{ik} v_k \right|.$$

Par inégalités triangulaires :

$$\begin{aligned}\|A \times v\|_\infty &\leqslant \max_{i<d} \sum_{k<d} |A_{ik}||v_k| \leqslant \max_{i<d} \sum_{k<d} (|A_{ik}| \max_{t<d} |v_t|) = \max_{t<d} |v_t| \left(\max_{i<d} \sum_{k<d} |A_{ik}| \right) \\ &= |||A|||_\infty \|v\|_\infty.\end{aligned}$$

□

La sous-multiplicativité est également vérifiée par le produit de deux matrices :

Lemme 6.2. Sous-multiplicativité de $|||\cdot|||_\infty$, produit matrice-matrice 
Soient $d \in \mathbb{N}^*$, $A, B \in \mathbb{R}^{d\times d}$. Alors :

$$|||A \times B|||_\infty \leqslant |||A|||_\infty |||B|||_\infty.$$

Démonstration. Par définition de $|||\cdot|||_\infty$ et du produit matrice-matrice :

$$|||A \times B|||_\infty = \max_{i<d} \sum_{j<d} |(A \times B)_{ij}| = \max_{i<d} \sum_{j<d} \left| \sum_{k<d} A_{ik} B_{kj} \right|.$$

Par inégalités triangulaires :

$$
\begin{aligned}
|||A \times B|||_\infty &\leqslant \max_{i<d} \sum_{j<d} \sum_{k<d} |A_{ik}||B_{kj}| = \max_{i<d} \sum_{k<d} \sum_{j<d} |A_{ik}||B_{kj}| \\
&= \max_{i<d} \sum_{k<d} |A_{ik}| \sum_{j<d} |B_{kj}| \leqslant \max_{i<d} \sum_{k<d} \left(|A_{ik}| \left(\max_{t<d} \sum_{j<d} |B_{tj}| \right) \right) \\
&= \max_{i<d} \left(\left(\max_{t<d} \sum_{j<d} |B_{tj}| \right) \sum_{k<d} |A_{ik}| \right) = \left(\max_{t<d} \sum_{j<d} |B_{tj}| \right) \max_{i<d} \sum_{k<d} |A_{ik}| = |||A|||_\infty |||B|||_\infty .
\end{aligned}
$$

□

Les démonstrations des lemmes 6.1 et 6.2 reposent sur des principes mathématiques assez simples. Cependant, les preuves formelles correspondantes s'avèrent relativement fastidieuses, incluant de nombreuses manipulations de grands opérateurs (échanges de sommes, multiplication par une constante dans les sommations, etc).

6.6 Erreurs d'arrondi d'opérations matricielles

6.6.1 Erreurs d'arrondi du produit scalaire

Les produits matrice-vecteur et matrice-matrice sont intrinsèquement liés à la notion de produit scalaire de vecteurs. Le produit scalaire de $a, b \in \mathbb{R}^d$ est défini comme :

$$a \cdot b = \sum_{i<d} a_i b_i .$$

Une fois encore, l'usage des grands opérateurs apparaît naturelle pour la formalisation du produit scalaire. Pour étudier les erreurs d'arrondi associées à l'implémentation du produit scalaire en arithmétique à virgule flottante, il s'avère utile de définir le produit scalaire « arrondi » par une fonction $\mathscr{W}$ pouvant être aussi bien l'identité qu'un arrondi en arithmétique à virgule flottante (dans le même esprit que pour la définition des implémentations de méthodes de Runge-Kutta en section 6.4.1).

```
Definition dot_product {d} (a b : 'cV[R]_d) (W : R → R) :=
        \big[(fun x y ⇒ W (x + y)) / 0]_(i < d) W ((a i) * (b i)).
```

Dans la définition du produit scalaire arrondi, l'opération sur laquelle itère le grand opérateur n'est pas nécessairement associative, ce qui est peu commun. En l'absence de loi d'associativité d'une opération élémentaire, un grand opérateur (voir section 2.2.3) itérant cette opération suppose un parenthésage de la droite vers la gauche. De manière cohérente, la notation mathématique $\bigoplus$ (grand opérateur associée à l'opération flottante élémentaire $\oplus$) suit cette convention, *i.e.* :

$$\bigoplus_{i<d} v_i = v_0 \oplus \left(v_1 \oplus \cdots \oplus \left(v_{d-2} \oplus v_{d-1} \right) \ldots \right).$$

C'est donc cette convention de parenthésage que nous choisissons d'imposer [2].

Le produit scalaire de deux vecteurs étant défini comme la somme des produits des composantes des deux vecteurs, nous commençons par borner l'erreur d'arrondi associée aux sommations. Le résultat dont nous avons besoin est une version affaiblie du théorème 2.10 pour un ordre fixé d'évaluation des sommations (de la droite vers la gauche). Ce résultat affaibli a été formellement prouvé en Coq par Boldo *et al.* [34]. Cependant, ces travaux n'utilisent ni les vecteurs de la bibliothèque Mathematical Components ni les grands opérateurs. La somme est définie *via* l'opérateur `fold_right` itérant l'addition sur une liste de nombres flottants. L'énoncé Coq correspondant est le suivant (*modulo* nos propres notations) :

```
Theorem error_sum_n: forall l: list R, Forall format l →
    let e:= fold_right Rplus 0 l in
    let f:= fold_right (fun x y ⇒ x (+) y) 0 l in
    let a:= fold_right Rplus 0 (map Rabs l) in
          Rabs (f-e) <= (INR (length l) -1)*u*a.
```

Nous avons adapté ce résultat en remplaçant la liste de valeurs par un vecteur de même taille et l'opérateur `fold_right` par un grand opérateur. Nous avons en outre remplacé l'unité d'arrondi u par la quantité $\frac{u}{1+u}$ en tirant parti de la borne raffinée proposée par Jeannerod et Rump pour les opérations élémentaires en arrondi au plus proche (lemme 2.4). L'énoncé Coq obtenu est [3] :

```
Theorem sum_n_error_bigop : forall {d} (v : 'cV_d.+1),
    format_mat v →
    let e:= \big[Rplus / 0]_(i < d.+1) (v i) in
    let f:= \big[(fun x y ⇒ x (+) y) / 0]_(i < d.+1) (v i) in
    let a:= \big[Rplus / 0]_(i < d.+1) Rabs (v i) in
         Rabs (f-e) <= INR d*(u/(1+u))*a.
```

Une borne sur l'erreur d'arrondi commise lors du calcul du produit scalaire de deux vecteurs peut être dérivée de la borne d'erreur de la sommation :

Lemme 6.3. Borne d'erreur d'arrondi du produit scalaire
Soient $d \in \mathbb{N}^*$. Soient $a, b \in \mathbb{R}^d$. Alors :

$$\left| \bigoplus_{i<d} a_i \otimes b_i - \sum_{i<d} a_i b_i \right| \leqslant du \sum_{i<d} |a_i||b_i| + (d^2u + d)\frac{\eta}{2}.$$

2. Les démonstrations montrant l'existence d'une borne d'erreur quel que soit l'ordre d'évaluation des opérations sont particulièrement techniques et difficiles à formaliser. C'est par exemple le cas du théorème 2.10 dont la preuve papier est donnée pour tout ordre d'évaluation par Jeannerod et Rump [119] mais que Boldo [34] a formalisé pour l'ordre d'évaluation de la droite vers la gauche.

3. Dans ce cas, le vecteur `v` est de type `'cV_d.+1`, ce qui permet d'ignorer le cas de la dimension 0.

Démonstration. D'après la borne sur l'erreur d'arrondi commise lors la sommation de nombres flottants (théorème 2.10), comme pour tout $0 \leqslant i < d$, $a_i \otimes b_i \in \mathbb{F}$:

$$\left| \bigoplus_{i<d} a_i \otimes b_i - \sum_{i<d} a_i \otimes b_i \right| \leqslant \frac{(d-1)u}{1+u} \sum_{i<d} |a_i \otimes b_i|.$$

Par inégalités triangulaires, puis *via* le lemme 2.4 et le fait que $\frac{u}{1+u} \leqslant u$:

$$\begin{aligned}\left| \bigoplus_{i<d} a_i \otimes b_i - \sum_{i<d} a_i b_i \right| &\leqslant \left| \bigoplus_{i<d} a_i \otimes b_i - \sum_{i<d} a_i \otimes b_i \right| + \left| \sum_{i<d} a_i \otimes b_i - \sum_{i<d} a_i b_i \right| \\ &\leqslant \frac{(d-1)u}{1+u} \sum_{i<d} |a_i \otimes b_i| + \sum_{i<d} \left(u|a_i||b_i| + \frac{\eta}{2} \right).\end{aligned}$$

Les lemmes 2.2 et 2.3 permettent d'obtenir :

$$\forall i, |a_i \otimes b_i| \leqslant (1+u)|a_i||b_i| + \frac{\eta}{2}.$$

Donc :

$$\begin{aligned}\left| \bigoplus_{i<d} a_i \otimes b_i - \sum_{i<d} a_i b_i \right| &\leqslant \frac{(d-1)u}{1+u} \sum_{i<d} \left((1+u)|a_i||b_i| + \frac{\eta}{2} \right) + \sum_{i<d} \left(u|a_i||b_i| + \frac{\eta}{2} \right) \\ &\leqslant (d-1)u \sum_{i<d} |a_i||b_i| + \frac{(d-1)u}{1+u} d\frac{\eta}{2} + u \sum_{i<d} |a_i||b_i| + d\frac{\eta}{2} \\ &\leqslant du \sum_{i<d} |a_i||b_i| + (d^2u + d)\frac{\eta}{2}.\end{aligned}$$

□

Comme nous l'avions évoqué dans le chapitre 3, un résultat similaire a été démontré par Roux [177, Lemma 3.11]. Cependant, ces travaux utilisent une spécification de l'arithmétique à virgule flottante plutôt que de se baser directement sur les formats flottants de Flocq. En outre, les résultats qui y sont formalisés sont basés sur la borne « standard » d'erreur pour les sommations (borne de Higham [107], voir théorème 2.9) alors que nous utilisons la borne raffinée qui ne fait pas intervenir la quantité γ_d (borne de Jeannerod et Rump [119], voir théorème 2.10).

6.6.2 Erreurs d'arrondi de produits matriciels

Dans cette section, nous nous intéressons aux erreurs d'arrondi induites par une implémentation en arithmétique à virgule flottante des produits matrice-vecteur et matrice-matrice. Borner ces erreurs constitue en effet une étape utile à l'étude des erreurs d'arrondi des méthodes de Runge-Kutta dans le cas de systèmes linéaires multidimensionnels.

Il convient tout d'abord de définir formellement les opérations matricielles arrondies. Comme pour la multiplication matricielle exacte dans Mathematical Components, nous définissons l'implémentation de la multiplication matricielle en arithmétique flottante pour toute paire de matrices, pas nécessairement carrées :

```
Definition mulmx_rnd {m n p}
  (A : 'M_(m, n)) (B : 'M_(n, p)) (W : R → R) : 'M[R]_(m, p) :=
    \matrix_(i, j) \big[(fun x y ⇒ W (x + y)) /0]_k W (A i k * B k j).
```

À partir des bornes d'erreurs d'arrondi du produit scalaire de vecteurs, il est possible de dériver une borne sur l'erreur d'arrondi du produit entre une matrice carrée et un vecteur colonne :

Théorème 6.4. Borne d'erreur d'arrondi du produit matrice-vecteur 
Soient $d \in \mathbb{N}^*, v \in \mathbb{R}^d, A \in \mathbb{R}^{d\times d}$. *Alors :*

$$\|A \otimes v - Av\|_\infty \leqslant du|||A|||_\infty \|v\|_\infty + (d^2u + d)\frac{\eta}{2}.$$

Démonstration. Par définition de $\otimes$ et de $\|\cdot\|_\infty$,

$$\|A \otimes v - Av\|_\infty = \max_{i<d} |(A \otimes v - Av)_i| = \max_{i<d} \left| \bigoplus_{k<d} A_{ik} \otimes v_k - \sum_{k<d} A_{ik}v_k \right|.$$

Donc, d'après le lemme 6.3 et après simplification :

$$\begin{aligned}\|A \otimes v - Av\|_\infty &\leqslant \max_{i<d} \left(du \sum_{k<d} |A_{ik}||v_k| + (d^2u + d)\frac{\eta}{2} \right) \\ &= du \max_{i<d} \left(\sum_{k<d} |A_{ik}||v_k| \right) + (d^2u + d)\frac{\eta}{2}.\end{aligned}$$

Donc :

$$\begin{aligned}\|A \otimes v - Av\|_\infty &\leqslant du \max_{i<d} \left(\sum_{k<d} |A_{ik}| \max_{t<d} |v_t| \right) + (d^2u + d)\frac{\eta}{2} \\ &\leqslant du \max_{t<d} |v_t| \max_{i<d} \left(\sum_{k<d} |A_{ik}| \right) + (d^2u + d)\frac{\eta}{2} \\ &= du\|v\|_\infty \max_{i<d} \left(\sum_{k<d} |A_{ik}| \right) + (d^2u + d)\frac{\eta}{2} \\ &= du|||A|||_\infty \|v\|_\infty + (d^2u + d)\frac{\eta}{2}.\end{aligned}$$

□

De même, il est possible de dériver une borne sur l'erreur d'arrondi du produit de deux matrices carrées.

Théorème 6.5. Borne d'erreur d'arrondi du produit matrice-matrice 
Soient $d \in \mathbb{N}^*, A, B \in \mathbb{R}^{d\times d}$. *Alors :*

$$|||A \otimes B - AB|||_\infty \leqslant du|||A|||_\infty |||B|||_\infty + d(d^2u + d)\frac{\eta}{2}.$$

Démonstration. Par définition de $\otimes$ et de $|||\cdot|||_\infty$,

$$|||A \otimes B - AB|||_\infty = \max_{i<d} \sum_{j<d} |(A \otimes B - AB)_{ij}| = \max_{i<d} \sum_{j<d} \left| \bigoplus_{k<d} A_{ik} \otimes B_{kj} - \sum_{k<d} A_{ik} B_{kj} \right|.$$

Donc, en appliquant le lemme 6.3 aux applications partielles $k \mapsto A_{ik}$ et $k \mapsto B_{kj}$:

$$\begin{aligned}
|||A \otimes B - AB|||_\infty &\leqslant \max_{i<d} \sum_{j<d} \left(du \sum_{k<d} |A_{ik}||B_{kj}| + (d^2u + d)\frac{\eta}{2} \right) \\
&\leqslant du \max_{i<d} \sum_{j<d} \left(\sum_{k<d} |A_{ik}||B_{kj}| \right) + d(d^2u + d)\frac{\eta}{2} \\
&= du \max_{i<d} \sum_{k<d} \left(\sum_{j<d} |A_{ik}||B_{kj}| \right) + d(d^2u + d)\frac{\eta}{2} \\
&= du \max_{i<d} \sum_{k<d} \left(|A_{ik}| \left(\sum_{j<d} |B_{kj}| \right) \right) + d(d^2u + d)\frac{\eta}{2} \\
&\leqslant du \max_{i<d} \sum_{k<d} \left(|A_{ik}| \left(\max_{t<d} \sum_{j<d} |B_{tj}| \right) \right) + d(d^2u + d)\frac{\eta}{2} \\
&= du \max_{i<d} \sum_{k<d} \left(|A_{ik}| |||B|||_\infty \right) + d(d^2u + d)\frac{\eta}{2} = du|||A|||_\infty |||B|||_\infty + d(d^2u + d)\frac{\eta}{2}.
\end{aligned}$$

□

6.6.3 Erreurs d'arrondi pour le calcul des coefficients

Pour borner les erreurs locales associées aux méthodes de Runge-Kutta classiques dans le cas unidimensionnel (voir section 5.4.2), nous avons dû borner les erreurs commises lors du calcul de coefficients scalaires dépendant de h et de λ, *e.g.* $\frac{h\lambda}{6}, \frac{h^2\lambda^2}{2}$, etc. Nous avions à cet effet utilisé l'outil Gappa, qui tire parti des intervalles de valeurs possibles pour le produit $h\lambda$, ces intervalles étant obtenus à partir de la stabilité linéaire.

Dans le cas multidimensionnel, les coefficients apparaissant dans l'implémentation « à la Horner » des méthodes de Runge-Kutta sont des matrices de la forme $\alpha\mathcal{A}$, avec $\alpha \in \mathbb{R}$ (*e.g.* h, $\frac{h}{2}$, etc) et $\mathcal{A} \in \mathbb{R}^{d\times d}$, *e.g.* A, A^2, etc. Le calcul de ces matrices en arithmétique à virgule flottante conduit à des résultats de la forme $\widetilde{\alpha} \otimes \widetilde{\mathcal{A}}$.

L'outil Gappa n'est plus directement utilisable pour exhiber une borne sur l'erreur d'arrondi $\widetilde{\alpha} \otimes \widetilde{\mathcal{A}} - \alpha\mathcal{A}$, qui dépend de la norme de $\mathcal{A}$.

Lemme 6.6. Borne sur l'erreur d'arrondi et calcul des coefficients 

Soient $d \in \mathbb{N}^*$, $\alpha \in \mathbb{R}$, $\widetilde{\alpha} \in \mathbb{F}$, $\mathcal{A} \in \mathbb{R}^{d\times d}$, $\widetilde{\mathcal{A}} \in \mathbb{F}^{d\times d}$, $B, C, D \in \mathbb{R}_+$. Supposons que :

- $|\widetilde{\alpha} - \alpha| \leqslant B$;
- $|||\widetilde{\mathcal{A}} - \mathcal{A}|||_\infty \leqslant Cu|||\mathcal{A}|||_\infty + D\eta$.

Alors :

$$|||\widetilde{\alpha}\otimes\widetilde{\mathcal{A}} - \alpha\mathcal{A}|||_\infty \leqslant \left((1+Cu)\left(u(B+|\alpha|)+B\right) + Cu|\alpha|\right)|||\mathcal{A}|||_\infty + \left((1+u)(B+|\alpha|D) + \frac{d}{2}\right)\eta.$$

Démonstration. Nous avons : $|||\widetilde{\alpha} \otimes \widetilde{\mathcal{A}} - \widetilde{\alpha}\widetilde{\mathcal{A}}|||_\infty = \max_{i<d} \sum_{j<d} |(\widetilde{\alpha} \otimes \widetilde{\mathcal{A}})_{ij} - (\widetilde{\alpha}\widetilde{\mathcal{A}})_{ij}|$,

donc, d'après les lemmes 2.2 et 2.3 :

$$|||\widetilde{\alpha} \otimes \widetilde{\mathcal{A}} - \widetilde{\alpha}\widetilde{\mathcal{A}}|||_\infty \leqslant \max_{i<d} \sum_{j<d} \left(u|\widetilde{\alpha}||\widetilde{\mathcal{A}}_{ij}| + \frac{\eta}{2} \right) \leqslant u|\widetilde{\alpha}||||\widetilde{\mathcal{A}}|||_\infty + \frac{d\eta}{2}.$$

Donc : $|||\widetilde{\alpha} \otimes \widetilde{\mathcal{A}} - \widetilde{\alpha}\widetilde{\mathcal{A}}|||_\infty \leqslant u\,(B + |\alpha|)\,|||\widetilde{\mathcal{A}}|||_\infty \leqslant u\,(B + |\alpha|)\,((1 + Cu)|||\mathcal{A}|||_\infty + D\eta) + \dfrac{d\eta}{2}$.

Par inégalités triangulaires, nous avons :

$$\begin{aligned}
|||\widetilde{\alpha} \otimes \widetilde{\mathcal{A}} - \alpha\mathcal{A}|||_\infty &\leqslant |||\widetilde{\alpha} \otimes \widetilde{\mathcal{A}} - \widetilde{\alpha}\widetilde{\mathcal{A}}|||_\infty + |||\widetilde{\alpha}\widetilde{\mathcal{A}} - \alpha\widetilde{\mathcal{A}}|||_\infty + |||\alpha\widetilde{\mathcal{A}} - \alpha\mathcal{A}|||_\infty + \frac{d\eta}{2} \\
&\leqslant u\,(B + |\alpha|)\,((1 + Cu)|||\mathcal{A}|||_\infty + D\eta) + |\widetilde{\alpha} - \alpha||||\widetilde{\mathcal{A}}|||_\infty + |\alpha||||\widetilde{\mathcal{A}} - \mathcal{A}|||_\infty + \frac{d\eta}{2} \\
&\leqslant u\,(B + |\alpha|)\,((1 + Cu)|||\mathcal{A}|||_\infty + D\eta) + B|||\widetilde{\mathcal{A}}|||_\infty + |\alpha|Cu|||\mathcal{A}|||_\infty + |\alpha|D\eta + \frac{d\eta}{2} \\
&\leqslant ((1 + Cu)\,(u(B + |\alpha|) + B) + Cu|\alpha|)\,|||\mathcal{A}|||_\infty + (1 + u)(B + |\alpha|)D\eta + \frac{d\eta}{2}.
\end{aligned}$$

□

Le lemme 6.6 sera utilisé en section 6.7.2 dans les étapes intermédiaires permettant de borner les erreurs locales des méthodes d'Euler et de Runge-Kutta 2.

Remarque. *Dans les implémentations « à la Horner », il n'y a pas de puissances explicites de la matrice A et ainsi $\mathcal{A} = A$ et $\widetilde{\mathcal{A}} = \widetilde{A}$ avec $\widetilde{A}$ la matrice A dont tous les coefficients ont été arrondis. En arrondi au plus proche, dans le format* `FLT`*, nous avons alors $C = 1$ et $D = \frac{d}{2}$, i.e. $|||\widetilde{A} - A|||_\infty \leqslant u|||A|||_\infty + \frac{d\eta}{2}$. Le lemme 6.6 permet cependant de gérer le cas $|||\widetilde{A}^n - A^n|||_\infty$ pour $n > 1$ mais la détermination de C et D s'avère plus délicate (voir [108, §4.2] pour plus de détails).*

6.7 Erreurs locales

6.7.1 Résultats génériques pour les erreurs locales

Pour borner les erreurs d'arrondi locales induites par l'implémentation de méthodes de Runge-Kutta appliquées à des systèmes matriciels, nous adoptons la même approche que dans le cas unidimensionnel (voir section 5.4.1). Nous démontrons en effet des lemmes tirant parti de la forme des implémentations. Ces résultats sont génériques et couvrent un nombre important d'implémentations possibles, parmi lesquelles nous trouvons les évaluations « à la Horner » présentées en section 6.2.2.

Nous commençons par la sommation de deux vecteurs flottants X_1 et X_2 correspondant à des multiples d'un même vecteur y (au sens du produit matrice-vecteur) :

Lemme 6.7. *Soient $d \in \mathbb{N}^*$, $y \in \mathbb{R}^d$, $C_1, C_2 \in \mathbb{R}_+$, $A_1, A_2 \in \mathbb{R}^{d\times d}$, $X_1, X_2 \in \mathbb{F}^d$ tels que :*

- $\|X_1 - A_1 y\|_\infty \leqslant C_1 u \|y\|_\infty + D_1\eta$;
- $\|X_2 - A_2 y\|_\infty \leqslant C_2 u \|y\|_\infty + D_2\eta$.

Alors :

$$\|X_1 \oplus X_2 - (A_1 + A_2)y\|_\infty \leqslant \\ (C_1 u + C_2 u + u(|||A_1|||_\infty + |||A_2|||_\infty + C_1 u + C_2 u))\,\|y\|_\infty + (1+u)(D_1+D_2)\eta.$$

Démonstration. Par inégalités triangulaires :

$$\begin{aligned}
&\|X_1 \oplus X_2 - (A_1 + A_2)y\|_\infty \leqslant \\
&\qquad \|X_1 \oplus X_2 - (X_1 + X_2)\|_\infty + \|X_1 - A_1 y\|_\infty + \|X_2 - A_2 y\|_\infty \\
&\qquad \leqslant u\|X_1 + X_2\|_\infty + C_1 u\|y\|_\infty + D_1\eta + C_2 u\|y\|_\infty + D_2\eta \\
&\qquad \leqslant u\|X_1\|_\infty + u\|X_2\|_\infty + C_1 u\|y\|_\infty + D_1\eta + C_2 u\|y\|_\infty + D_2\eta \\
&\qquad \leqslant u\left(\|A_1 y\|_\infty + C_1 u\|y\|_\infty + D_1\eta\right) + u\left(\|A_2 y\|_\infty + C_2 u\|y\|_\infty + D_2\eta\right) \\
&\qquad\quad + C_1 u\|y\|_\infty + D_1\eta + C_2 u\|y\|_\infty + D_2\eta.
\end{aligned}$$

Donc, en appliquant le lemme 6.1 (sous-multiplicativité) :

$$\begin{aligned}
&\|X_1 \oplus X_2 - (A_1 + A_2)y\|_\infty \\
&\quad \leqslant u\left(|||A_1|||_\infty\|y\|_\infty + C_1 u\|y\|_\infty + D_1\eta\right) + u\left(|||A_2|||_\infty\|y\|_\infty + C_2 u\|y\|_\infty + D_2\eta\right) \\
&\qquad + C_1 u\|y\|_\infty + D_1\eta + C_2 u\|y\|_\infty + D_2\eta \\
&\quad = (C_1 u + C_2 u + u(|||A_1|||_\infty + |||A_2|||_\infty + C_1 u + C_2 u))\,\|y\|_\infty + (1+u)(D_1+D_2)\eta.
\end{aligned}$$

□

La démonstration formelle de ce résultat pose relativement peu de difficultés. En particulier, elle repose sur un nombre restreint d'éléments de calcul matriciel.

Nous proposons ensuite un résultat qui généralise le lemme 5.4 au cas multidimensionnel et couvre notamment les implémentations « à la Horner » (voir section 6.2.2).

Lemme 6.8. *Soient $d \in \mathbb{N}^*$, $y \in \mathbb{R}^n$, $C_1, C_2, C_3, D_1, D_2, D_3 \in \mathbb{R}_+$, $A_1, A_2, A_3 \in \mathbb{R}^{d\times d}$. Soient $X_1, X_3 \in \mathbb{F}^d$, $\widetilde{A_2} \in \mathbb{F}^{d\times d}$. Supposons que :*

- $\|X_1 - A_1 y\|_\infty \leqslant C_1 u\|y\|_\infty + D_1\eta$;
- $|||\widetilde{A_2} - A_2|||_\infty \leqslant C_2 u + D_2\eta$;
- $\|X_3 - A_3 y\|_\infty \leqslant C_3 u\|y\|_\infty + D_3\eta$.

Alors :

$$\begin{aligned}
&\|X_1 \oplus \widetilde{A_2} \otimes X_3 - (A_1 + A_2 A_3)y\|_\infty \leqslant \\
&\Big((1+u)\Big(C_1 u + (1+du)(C_2 u|||A_3|||_\infty + C_3 u|||A_2|||_\infty + C_2 C_3 u^2 + (C_3 u + |||A_3|||_\infty)D_2\eta\Big)\Big) \\
&\qquad + u|||A_1|||_\infty + ((d+1)u + du^2)|||A_2|||_\infty|||A_3|||_\infty\Big)\,\|y\|_\infty \\
&\qquad + (1+u)\left(D_1 + (1+du)(C_2 u + D_2\eta + |||A_2|||_\infty)D_3 + \frac{1}{2}(d^2 u + d)\right)\eta.
\end{aligned}$$

Démonstration. Par inégalités triangulaires, puis *via* plusieurs applications du lemme 6.1 :

$$\begin{aligned}\|\widetilde{A_2} \otimes X_3 - A_2A_3y\|_\infty &\leqslant \|\widetilde{A_2} \otimes X_3 - \widetilde{A_2}A_3y\|_\infty + \|\widetilde{A_2}A_3y - A_2A_3y\|_\infty \\ &\leqslant \|\widetilde{A_2} \otimes X_3 - \widetilde{A_2}X_3\|_\infty + \|\widetilde{A_2}X_3 - \widetilde{A_2}A_3y\|_\infty + |||\widetilde{A_2} - A_2|||_\infty \|A_3y\|_\infty \\ &\leqslant \|\widetilde{A_2} \otimes X_3 - \widetilde{A_2}X_3\|_\infty + |||\widetilde{A_2}|||_\infty \|X_3 - A_3y\|_\infty + |||\widetilde{A_2} - A_2|||_\infty |||A_3|||_\infty \|y\|_\infty.\end{aligned}$$

En utilisant les hypothèses du lemme et en appliquant le théorème 6.4 au produit de $\widetilde{A_2}$ et X_3 :

$$\begin{aligned}&\|\widetilde{A_2} \otimes X_3 - A_2A_3y\|_\infty \\ &\quad\leqslant du|||\widetilde{A_2}|||_\infty \|X_3\|_\infty + (d^2u + d)\frac{\eta}{2} + |||\widetilde{A_2}|||_\infty \|X_3 - A_3y\|_\infty \\ &\qquad + (C_2u + D_2\eta)|||A_3|||_\infty \|y\|_\infty \\ &\quad\leqslant du(C_2u + D_2\eta + |||A_2|||_\infty)((C_3u + |||A_3|||_\infty)\|y\|_\infty + D_3\eta) + (d^2u + d)\frac{\eta}{2} \\ &\qquad + (C_2u + D_2\eta + |||A_2|||_\infty)(C_3u\|y\|_\infty + D_3\eta) + (C_2u + D_2\eta)|||A_3|||_\infty \|y\|_\infty.\end{aligned}$$

Donc :

$$\begin{aligned}&\|\widetilde{A_2} \otimes X_3 - A_2A_3y\|_\infty \leqslant \\ &\qquad \Big((1 + du)\big(C_2u|||A_3|||_\infty + C_3u|||A_2|||_\infty + C_2C_3u^2 \\ &\qquad\qquad +D_2(|||A_3|||_\infty + C_3u)\eta) + du|||A_2|||_\infty|||A_3|||_\infty\big)\|y\|_\infty \\ &\qquad\quad + \left((1 + du)(C_2u + D_2\eta + |||A_2|||_\infty)D_3 + \frac{1}{2}(d^2u + d)\right)\eta\end{aligned}$$

Donc, par application du lemme 6.7 avec X_1 et A_1 restant inchangés, $\widetilde{A_2} \otimes X_3$ identifié à X_2 (du lemme 6.7) et A_2A_3 identifié à A_2 (du lemme 6.7) :

$$\begin{aligned}&\|X_1 \oplus \widetilde{A_2} \otimes X_3 - (A_1 + A_2A_3)y\|_\infty \leqslant \\ &\quad (u(|||A_1|||_\infty + |||A_2A_3|||_\infty) \\ &\qquad + (1 + u)\big(C_1u + (1 + du)\big(C_2u|||A_3|||_\infty + C_3u|||A_2|||_\infty + C_2C_3u^2 \\ &\qquad\qquad +D_2(|||A_3|||_\infty + C_3u)\eta) + du|||A_2|||_\infty|||A_3|||_\infty)\big)\|y\|_\infty \\ &\qquad\quad + (1 + u)\left(D_1 + (1 + du)(C_2u + D_2\eta + |||A_2|||_\infty)D_3 + \frac{1}{2}(d^2u + d)\right)\eta \\ &\leqslant \big((1 + u)\big(C_1u + (1 + du)(C_2u|||A_3|||_\infty + C_3u|||A_2|||_\infty + C_2C_3u^2 + (C_3u + |||A_3|||_\infty)D_2\eta\big)\big) \\ &\qquad +u|||A_1|||_\infty + ((d + 1)u + du^2)|||A_2|||_\infty|||A_3|||_\infty\big)\|y\|_\infty \\ &\qquad + (1 + u)\left(D_1 + (1 + du)(C_2u + D_2\eta + |||A_2|||_\infty)D_3 + \frac{1}{2}(d^2u + d)\right)\eta.\end{aligned}$$

□

Lorsque les dépassements de capacité inférieurs ne sont pas pris en compte, le terme en η est nul. Notons qu'à la différence du lemme 5.4 utilisé dans le cas des systèmes linéaires scalaires, le lemme 6.8 ne permet pas de construire une valeur seuil M à partir

de laquelle les dépassements inférieurs ne contribuent plus à la borne d'erreur. En effet, nous ne faisons ici aucune hypothèse sur la norme de la matrice A. De plus, même si nous supposons que la valeur $|||A|||_\infty$ est supérieure à une certaine valeur, cela ne donne aucune information quant aux potentiels dépassements inférieurs au cours de calculs intermédiaires d'opérations matricielles. Dans ce cadre, nous estimons qu'une analyse par composantes aurait pu s'avérer utile.

Avant d'instancier le lemme 6.8 à des méthodes de Runge-Kutta classiques, nous le simplifions en supposant que $du < 0{,}01$. À condition que la dimension d du système ne soit pas démesurément grande, cette hypothèse s'avère en effet raisonnable pour la plupart des formats flottants utilisés en pratique. Par exemple, dans le cas du format *binary64*, $u = 2^{-53}$ et l'hypothèse $du < 0{,}01$ est vérifiée pour tout $d \leqslant 0{,}01 \times 2^{53} \simeq 9 \times 10^{13}$. En *binary32*, $u = 2^{-24}$ et l'hypothèse $du < 0{,}01$ est vérifiée pour tout $d \leqslant 0{,}01 \times 2^{24} \simeq 167\ 772$. En pratique, peu d'applications utilisent des matrices de taille supérieure à 10^4, ce nombre pouvant dans de rares cas atteindre 10^6 (dans ce cas, notre approche reste valable pour les nombres flottants en double précision). La version simplifiée du lemme 6.8 s'énonce comme suit :

Corollaire 6.9.  *Soient $d \in \mathbb{N}^*$, $y \in \mathbb{R}^n$, $C_1, C_2, C_3, D_1, D_2, D_3 \in \mathbb{R}_+$, $A_1, A_2, A_3 \in \mathbb{R}^{d\times d}$. Soient $X_1, X_3 \in \mathbb{F}^d$, $\widetilde{A_2} \in \mathbb{F}^{d\times d}$. Supposons que :*

- $\|X_1 - A_1 y\|_\infty \leqslant C_1 u \|y\|_\infty + D_1 \eta$;
- $|||\widetilde{A_2} - A_2|||_\infty \leqslant C_2 u + D_2 \eta$;
- $\|X_3 - A_3 y\|_\infty \leqslant C_3 u \|y\|_\infty + D_3 \eta$;
- $du < 0{,}01$.

Alors :

$$\begin{aligned}
&\|X_1 \oplus (\widetilde{A_2} \otimes X_3) - (A_1 + A_2 A_3) y\|_\infty \leqslant \\
&\Big(1{,}03\big(C_1 u + C_2 u |||A_3|||_\infty + C_3 u |||A_2|||_\infty + C_2 C_3 u^2 + (C_3 u + |||A_3|||_\infty) D_2 \eta\big) \\
&\quad + u|||A_1||| + (d + 1{,}01) u |||A_2|||_\infty |||A_3|||_\infty\Big) \|y\|_\infty \\
&\quad + \big(1{,}03(D_1 + (C_2 u + D_2 + |||A_2|||_\infty) D_3) + 0{,}6d\big)\, \eta.
\end{aligned}$$

La démonstration du corollaire 6.9 est essentiellement calculatoire et ne pose aucune difficulté particulière. Nous majorons notamment le terme $D_2\eta$ par D_2 dans le terme d'*underflow*, ce qui revient à majorer η^2 par η. Ce corollaire sera appliqué dans la section suivante pour borner les erreurs locales de méthodes classiques.

6.7.2 Bornes sur l'erreur locale de la méthode d'Euler

Dans cette section, nous instancions les résultats génériques de la section 6.7.1 et bornons les erreurs locales associées à la méthodes d'Euler. Le format flottant considéré est le format `FLT` de Flocq et les résultats sont paramétrés par la base et la précision de calcul. Sur la donnée de la dimension du système, la seule condition que doit satisfaire le format flottant est $du < 0{,}01$.

Lemme 6.10. Borne sur l'erreur locale pour la méthode d'Euler 
Soit $h \in \mathbb{F}$, $A \in \mathbb{R}^{d\times d}$. Soient $n \in \mathbb{N}$, $d \in \mathbb{N}^*$. Soit $u = \beta^{-p}$. Supposons que $du < 0{,}01$. Soit $(\widetilde{y_n})_{n\in\mathbb{N}}$ la suite des solutions calculées via une implémentation flottante « à la Horner » de la méthode d'Euler (voir exemple 6.3). Soit $(\varepsilon_n)_{n\in\mathbb{N}}$ la suite des erreurs locales associées à la méthode d'Euler. Alors :

$$\varepsilon_{n+1} \leqslant \left((1 + (d+3{,}1)\, h|||A|||_\infty)\, u + 0{,}59(1+h)d\eta\right)\ \|\widetilde{y_n}\|_\infty + 0{,}6d\eta.$$

Démonstration. Par définition de l'erreur locale et de la méthode d'Euler :

$$\varepsilon_{n+1} = \|(\widetilde{y_n} \oplus ((h \otimes \widetilde{A}) \otimes \widetilde{y_n})) - (I + hA \times I)\widetilde{y_n}\|_\infty.$$

La démonstration repose sur une unique application du corollaire 6.9 avec $y = \widetilde{y_n}, X_1 = \widetilde{y_n}, A_1 = I, \widetilde{A_2} = h \otimes \widetilde{A}, A_2 = hA, X_3 = \widetilde{y_n}$ et $A_3 = I$. Il faut donc exhiber les constantes C_1, C_2, C_3, D_1, D_2 et D_3.

Vérification des hypothèses.

- $\|X_1 - A_1 y\|_\infty = \|\widetilde{y_n} - I\widetilde{y_n}\|_\infty \leqslant 0\|\widetilde{y_n}\|_\infty + 0\eta$. Donc $C_1 = 0$ et $D_1 = 0$ sont valides.
- $|||\widetilde{A_2} - A_2|||_\infty = |||h \otimes \widetilde{A} - hA|||_\infty$.

  Donc, d'après le lemme 6.6 (avec $B = 0$, $C = d$, $D = \frac{d}{2}$) puis en utilisant l'hypothèse $du < 0{,}01$:

$$\begin{aligned}|||\widetilde{A_2} - A_2|||_\infty &\leqslant ((1+u)uh + uh)\,|||A|||_\infty + \left((1+u)h\frac{d}{2} + \frac{d}{2}\right)\eta \\ &\leqslant ((2+u)uh)|||A|||_\infty + 0{,}6(1+h)d\eta \\ &\leqslant 2{,}01h|||A|||_\infty u + 0{,}6(1+h)d\eta.\end{aligned}$$

  Donc $C_2 = 2{,}01h|||A|||_\infty$ et $D_2 = 0{,}6(1+h)d$ sont valides.
- $\|X_3 - A_3 y\|_\infty = \|\widetilde{y_n} - I\widetilde{y_n}\|_\infty \leqslant 0\|\widetilde{y_n}\|_\infty + 0\eta$. Donc $C_3 = 0$ et $D_3 = 0$ sont valides.

Application du corollaire 6.9. En appliquant le corollaire 6.9 puis en simplifiant :

$$\varepsilon_{n+1} \leqslant \left((1 + (d+3{,}1)\, h|||A|||_\infty)\, u + 0{,}59(1+h)d\eta\right)\ \|\widetilde{y_n}\|_\infty + 0{,}6d\eta.$$

□

6.7.3 Bornes sur l'erreur locale de la méthode RK2

Nous proposons une seconde application de l'analyse d'erreurs dans le cas de la méthode de Runge-Kutta d'ordre 2.

Lemme 6.11. Borne sur l'erreur locale pour la méthode RK2 

Soit $h \in \mathbb{F}$, $A \in \mathbb{R}^{d\times d}$. Soient $n \in \mathbb{N}$, $d \in \mathbb{N}^*$. Soit $u = \beta^{-p}$. Supposons que $du < 0{,}01$. Soit $(\widetilde{y_n})_{n\in\mathbb{N}}$ la suite des solutions calculées via une implémentation flottante « à la Horner » de la méthode RK2 (voir exemple 6.4). Soit $(\varepsilon_n)_{n\in\mathbb{N}}$ la suite des erreurs locales associées à la méthode de RK2. Alors :

$$\begin{aligned}\varepsilon_{n+1} \leqslant &\left(\left((d+5)hu + (6d+0{,}12d^2)\eta\right)|||A|||_\infty + \left((2d+6{,}6)hu+\eta\right)|||A|||_\infty^2\right.\\ &\left.+\left(2{,}4d+2{,}02d^2\right)\eta\right)\|\widetilde{y_n}\|_\infty + \left((0{,}6+h|||A|||_\infty)d+0{,}7d^2\right)\eta.\end{aligned}$$

Démonstration. La démonstration formelle repose sur plusieurs applications du corollaire 6.9, qui permettent de reconstruire la borne sur l'erreur locale de la méthode RK2. Les tableaux 6.1 et 6.2 contiennent les valeurs successives correspondant à C et D.

Dans un premier temps, nous nous intéressons à la sous-expression $\circ\left[\widetilde{y_n} + \frac{h\widetilde{A}}{2}\widetilde{y_n}\right]$, dont l'analyse d'erreurs est synthétisée dans le tableau 6.1. Les trois premières lignes correspondent aux bornes d'erreur sur $X_1 = \widetilde{y_n}$, $\widetilde{A_2} = h \oslash 2 \otimes \widetilde{A}$ et $X_3 = \widetilde{y_n}$. Les constantes C_1, C_3, D_1 et D_3 (première et troisième lignes) sont exhibées sans difficulté particulière. Pour obtenir C_2 (deuxième ligne), nous appliquons le lemme 6.6. Enfin, la quatrième ligne est obtenue par application du corollaire 6.9. Dans la colonne Cu, nous ne donnons pas explicitement la valeur de C, *e.g.* $1{,}2d\eta$ correspond à $C = 1{,}2d\xi$ car $1{,}2d\eta = 1{,}2d\xi u$

Tableau 6.1 – Sous-étape pour borner l'erreur locale de la méthode RK2

Terme FP	Matrice	Cu	D
$\widetilde{y_n}$	I	$C_1 u = 0$	$D_1 = 0$
$h \oslash 2 \otimes \widetilde{A}$	$\frac{hA}{2}$	$C_2 u = 1{,}6h\|\|\|A\|\|\|_\infty u + 0{,}6\eta$	$D_2 = 1{,}52d$
$\widetilde{y_n}$	I	$C_3 u = 0$	$D_3 = 0$
$\circ\left[\widetilde{y_n} + \frac{h\widetilde{A}}{2}\widetilde{y_n}\right]$	$I + \frac{hA}{2}$	$u + \left((1{,}7 + 0{,}6(d+1)h)u + 0{,}7\eta\right)\|\|\|A\|\|\|_\infty + 1{,}2d\eta$	$0{,}6d$

Ensuite, dans le tableau 6.2, nous reconstruisons la borne d'erreur associée à l'expression entière. Les trois premières lignes correspondent aux bornes d'erreur sur $X_1 = \widetilde{y_n}$, $\widetilde{A_2} = h \oslash 2 \otimes \widetilde{A}$ et $X_3 = \widetilde{y_n}$. Les constantes C_1 et D_1 sont obtenues trivialement, la constante C_2 est obtenue par application du lemme 6.6 et les constantes C_3 et D_3 correspondent au résultat final exhibé dans le tableau 6.1. Le résultat de la quatrième ligne est obtenu par application du corollaire 6.9. Nous noterons $C_{|||A|||} = (d+5)hu + (6d+0{,}12d^2)\eta$, $C_{|||A|||^2} = (2d+6{,}6)hu + \eta$ et $C_\eta = 2{,}4d + 2{,}02d^2$.

Tableau 6.2 – Étapes pour borner l'erreur locale de la méthode RK2

Terme FP	Matrice	Cu	D
$\widetilde{y_n}$	I	$C_1 u = 0$	$D_1 = 0$
$h \otimes \widetilde{A}$	hA	$C_2 u = 2{,}01h\|\|\|A\|\|\|_\infty u$	$D_2 = 0{,}6(1+h)d$
$\circ\left[\widetilde{y_n} + \frac{h\widetilde{A}}{2}\widetilde{y_n}\right]$	$I + \frac{hA}{2}$	$C_3 u = u + \left((1{,}7 + 0{,}6(d+1)h)u + 0{,}7\eta\right)\|\|\|A\|\|\|_\infty + 1{,}2d\eta$	$D_3 = 0{,}6d$
$\circ\left[\widetilde{y_n} + h\widetilde{A}\left(\widetilde{y_n} + \frac{h\widetilde{A}}{2}\widetilde{y_n}\right)\right]$	$I + hA\left(I + \frac{hA}{2}\right)$	$C_{\|\|\|A\|\|\|}\|\|\|A\|\|\|_\infty + C_{\|\|\|A\|\|\|^2}\|\|\|A\|\|\|_\infty^2 + C_\eta\eta$	$(0{,}6 + h\|\|\|A\|\|\|_\infty)d + 0{,}7d^2$

□

Automatisations et simplifications :

Les bornes obtenues s'avèrent lisibles (voir lemmes 6.10 et 6.11) et ne font pas apparaître de termes d'ordre supérieur (en u^2, etc). Nous avons majoré ces termes à chaque étape de la construction de la borne d'erreur en utilisant la tactique automatique `interval` [150] sous l'hypothèse $du < 0{,}01$ et la majoration $u^2 \leqslant 0{,}01u$ qui en découle.

Nous avons par ailleurs utilisé la tactique `eapply` de Coq afin d'appliquer mécaniquement le lemme 6.9 à chaque étape de la construction de la borne d'erreur locale sans avoir à donner explicitement les paramètres y, X_1, $\widetilde{A_2}$, X_3, A_1, A_2 et A_3. Il ne reste alors qu'à instancier manuellement les valeurs de C_1, C_2, C_3, D_1 et D_3 puis à montrer que les hypothèses du lemme 6.9 sont vérifiées.

6.8 Erreurs globales

La méthodologie permettant de passer des bornes d'erreurs locales aux bornes d'erreurs globales est empruntée à celle que nous appliquons dans le cas scalaire (voir section 5.5). En section 6.8.1, nous prouvons un théorème générique permettant de construire une borne sur l'erreur globale à partir des bornes sur les erreurs locales. Ensuite, en section 6.8.2, nous instancions ce résultat aux méthodes d'Euler et de Runge-Kutta 2 en utilisant les bornes d'erreurs locales exhibées en section 6.6.3.

6.8.1 Passage des erreurs locales aux erreurs globales

Nous bornons l'erreur d'arrondi globale induite par l'implémentation de méthodes de Runge-Kutta explicites appliquées à un système matriciel en fonction d'une borne sur les erreurs locales des itérations précédentes. Le résultat proposé est très proche du théorème 5.9, qui joue un rôle analogue pour le cas linéaire scalaire.

Théorème 6.12. Passage des erreurs locales aux erreurs globales 
Soit $h \in \mathbb{F}$, $A \in \mathbb{R}^{d\times d}$. Soit $C > 0$, $D \geqslant 0$. Posons $\kappa = \max(Cu + |||R(h,A)|||_\infty, 1)$. Supposons que pour tout $n \in \mathbb{N}^*, \varepsilon_n \leqslant Cu\|\widetilde{y_{n-1}}\|_\infty + D\eta$. Alors :

$$\forall n, E_n \leqslant (Cu + |||R(h,A)|||_\infty)^n \left(\varepsilon_0 + \frac{nCu\|y_0\|_\infty}{Cu + |||R(h,A)|||_\infty}\right) + n \times \kappa^n \times D\eta.$$

La démonstration du théorème 6.12 est très proche de la démonstration du théorème 5.9, si bien que nous ne redonnons pas les détails du raisonnement.

Nous n'avons jusqu'ici émis aucune hypothèse de stabilité linéaire, contrairement à ce qui est fait dans le cas scalaire (chapitre 5). Cela justifie l'introduction du terme κ, qui permet d'obtenir un résultat valable pour n'importe quels systèmes linéaires multidimensionnels. Lorsque $Cu + |||R(h,A)|||_\infty \leqslant 1$, nous avons également $\kappa \leqslant 1$ et le terme $n \times \kappa^n \times D\eta$ peut être majoré par $nD\eta$ de façon analogue à ce qui est fait dans le théorème 5.9. L'hypothèse de stabilité linéaire pour les systèmes matriciels réels est plus délicate à définir que dans le cas scalaire. Comme expliqué brièvement en

section 2.3.4, l'idée est d'imposer la condition de stabilité $|R(h,\lambda_i)| \leqslant 1$ sur chacune des valeurs propres λ_i de la matrice $R(h,A)$. Ce concept de stabilité est cependant difficilement exploitable dans une analyse par normes. Si nous nous limitons au cas où $|||R(h,A)|||_\infty < 1$, le théorème 6.12 assure que les bornes d'erreurs d'arrondi sont quasi-linéaires en le nombre d'itérations (car, en général, $Cu \ll |||R(h,A)|||_\infty$).

6.8.2 Bornes sur l'erreur globale de méthodes de Runge-Kutta

À partir des bornes d'erreurs locales exhibées en section 6.7.2 pour les méthodes d'Euler et RK2 puis en appliquant mécaniquement le théorème 6.12, nous bornons les erreurs globales de ces deux méthodes numériques classiques. Les résultats tiennent compte d'éventuels dépassements inférieurs.

Pour la méthode d'Euler, nous instancions le théorème 6.12 avec les constantes du lemme 6.10 et obtenons :

Théorème 6.13. Borne sur l'erreur globale de la méthode d'Euler 

Soit $h \in \mathbb{F}$, $A \in \mathbb{R}^{d\times d}$. *Soient* $n \in \mathbb{N}$, $d \in \mathbb{N}^*$. *Soit* $u = \beta^{-p}$. *Supposons que* $du < 0{,}01$. *Soit* E_n *l'erreur globale associée à la méthode d'Euler après* n *itérations, suivant une implémentation « à la Horner ». Posons* $Cu = (1 + (d+3{,}1)\,h|||A|||_\infty)\,u + 0{,}59(1+h)d\eta$, $D = 0{,}6d$ *et* $\kappa = \max(Cu + |||I + hA|||_\infty, 1)$. *Alors :*

$$E_n \leqslant (Cu + |||I + hA|||_\infty)^n \left(\varepsilon_0 + \frac{nCu\|y_0\|_\infty}{Cu + |||I + hA|||_\infty}\right) + n \times \kappa^n \times D\eta.$$

De même, pour la méthode de Runge-Kutta d'ordre 2, il suffit d'instancier le théorème 6.12 en utilisant les constantes données par le lemme 6.11.

Théorème 6.14. Borne sur l'erreur globale de la méthode RK2 

Soit $h \in \mathbb{F}$, $A \in \mathbb{R}^{d\times d}$. *Soient* $n \in \mathbb{N}$, $d \in \mathbb{N}^*$. *Soit* $u = \beta^{-p}$. *Supposons que* $du < 0{,}01$. *Soit* E_n *l'erreur globale associée à la méthode RK2 après* n *itérations, suivant une implémentation « à la Horner ». Posons* $Cu = ((d+5)hu + (6d + 0{,}12d^2)\eta)\,|||A|||_\infty + ((2d+6{,}6)hu + \eta)\,|||A|||_\infty^2 + (2{,}4d + 2{,}02d^2)\,\eta$, $D = (0{,}6 + h|||A|||_\infty)d + 0{,}7d^2$. *Posons également* $\kappa = \max\left(Cu + \left|\left|\left|I + hA + \frac{h^2A^2}{2}\right|\right|\right|_\infty, 1\right)$. *Alors :*

$$E_n \leqslant \left(Cu + \left|\left|\left|I + hA + \frac{h^2A^2}{2}\right|\right|\right|_\infty\right)^n \left(\varepsilon_0 + \frac{nCu\|y_0\|_\infty}{Cu + \left|\left|\left|I + hA + \frac{h^2A^2}{2}\right|\right|\right|_\infty}\right) + n \times \kappa^n \times D\eta.$$

6.9 Conclusion

Dans ce chapitre, nous avons proposé une méthodologie d'analyse générique permettant de borner l'accumulation d'erreurs d'arrondi dans l'implémentation en arithmétique à virgule flottante de méthodes de Runge-Kutta appliquées à des problèmes matriciels. Nous avons en outre formalisé l'ensemble des résultats dans l'assistant de

preuves Coq, ce qui permet d'augmenter la confiance accordée aux démonstrations des résultats présentés, souvent techniques et fastidieuses.

Nous avons choisi d'adopter une approche par normes, plus simple à utiliser et adaptée aux besoins d'une analyse dans le pire cas. La norme choisie est la norme $\|\cdot\|_\infty$ de Hölder qui mesure la plus grande composante (au sens de la valeur absolue) d'un vecteur et permet par conséquent de majorer l'ensemble des composantes du vecteur d'erreur. Nous avons formalisé la norme $\|\cdot\|_\infty$ ainsi que la norme matricielle subordonnée qui lui est associée. Nous avons prouvé les propriétés fondamentales de ces normes, notamment la propriété de sous-multiplicativité.

Dans le but de formaliser l'analyse d'erreurs d'arrondi des méthodes de Runge-Kutta, nous avons commencé par borner les erreurs d'arrondi associées aux produits matrice-vecteur et matrice-matrice en Coq. Les bornes exhibées sont basées sur des travaux récents de Jeannerod et Rump [119], qui ont proposé, pour des algorithmes de sommation, des bornes d'erreur d'arrondi plus fines que celles des ouvrages de référence tels que le livre de Higham [107].

Nous avons proposé une formalisation des méthodes de Runge-Kutta appliquées aux systèmes matriciels ainsi que leur implémentation et les erreurs d'arrondi locales et globales associées. Nous avons démontré formellement un ensemble de résultats génériques permettant de borner ces erreurs d'arrondi de façon mécanique. Ces résultats restent valides si nous tenons compte de potentiels dépassements de capacité inférieurs. Les dépassements de capacité supérieurs ne sont en revanche pas gérés, bien qu'il soit possible de les détecter *a posteriori* lorsque les valeurs exceptionnelles ∞ ou NaN sont produites. Nous avons ensuite instancié la méthodologie à deux méthodes classiques, *i.e.* les méthodes d'Euler et RK2.

La formalisation est basée sur la bibliothèque Flocq d'arithmétique des ordinateurs [37, 38] et sur les matrices de la bibliothèque Mathematical Components [144]. Elle comporte près de 5000 lignes de code Coq.

Les travaux de formalisation situés à l'intersection de l'arithmétique à virgule flottante et de l'analyse numérique matricielle sont relativement rares. À notre connaissance, les seuls travaux comparables sont ceux de Roux [177], qui propose une analyse d'erreurs d'arrondi en Coq pour plusieurs algorithmes numériques impliquant des opérations matricielles, comme la décomposition de Cholesky. Le chapitre 3 contient une comparaison entre l'approche de Roux et celle que nous adoptons.

Troisième partie

Vers la méthode des éléments finis : formalisation de résultats d'analyse fonctionnelle

Chapitre 7

Espaces de Hilbert et analyse fonctionnelle

Les équations aux dérivées partielles permettent de caractériser le comportement de phénomènes physiques ou biologiques très complexes, comme l'écoulement du sang dans une artère [198] ou de l'air sur le profil d'un avion [127]. Leur étude est donc fondamentale dans de nombreux domaines scientifiques, comme la mécanique des fluides, l'aéronautique, la médecine, la finance ou la météorologie. Il est généralement impossible d'exhiber la solution exacte d'une telle équation, notamment lorsque l'espace des solutions est irrégulier ou de dimension infinie. Comme dans le cas des équations différentielles ordinaires, des méthodes numériques permettent de calculer une solution approchée au problème. Les méthodes des volumes finis [67], des différences finies [189] et des éléments finis [49] sont les plus couramment utilisées.

La méthode des éléments finis consiste à choisir un maillage approchant le domaine d'intégration de l'équation aux dérivées partielles (le volume irrégulier et continu, *e.g.* l'artère ou le profil d'aile d'avion). Ce maillage est généralement constitué de polyèdres réguliers, dont la forme dépend du domaine d'intégration et appelés *éléments finis*. L'équation différentielle est linéarisée sur chacun de ces éléments finis, ce qui permet d'obtenir un ensemble d'équations linéaires. Ces équations sont, dans la plupart des cas, des équations différentielles ordinaires linéaires matricielles. Elles pourront être résolues à l'aide de méthodes de Runge-Kutta, dont les erreurs d'arrondi sont étudiées dans les chapitres 5 et 6 de ce document.

Notre souhait à long terme serait de prouver formellement des programmes implémentant ou utilisant la méthode des éléments finis, comme la bibliothèque C++ FELiScE [1] appliquée au domaine médical. Un tel but ne peut être atteint que si nous sommes capables de vérifier la correction de la méthode des éléments finis, c'est-à-dire de vérifier qu'elle permet de calculer une bonne approximation de la solution exacte. Or, la correction de la méthode repose sur des fondements conséquents d'analyse fonctionnelle [49, 64, 206]. Il s'agit de caractériser le domaine d'intégration *via* le concept

1. Finite Elements for Life Sciences and Engineering
`https://gforge.inria.fr/projects/felisce/`

d'espace de Hilbert et de définir le maillage comme sous-espace de dimension finie de cet espace de Hilbert. L'existence des solutions sur l'espace de Hilbert et sur le sous-espace de dimension finie, ainsi qu'une borne sur leur différence, sont assurées par le théorème de Lax–Milgram et le lemme de Céa. Dans le présent chapitre, ainsi que dans le chapitre 8, nous présentons une formalisation de ces fondements mathématiques. Ce travail est guidé par une preuve papier détaillée du théorème de Lax–Milgram, rédigée par Clément et Martin dans l'objectif d'une vérification formelle [50]. Ces travaux de formalisation sont basés sur la bibliothèque Coquelicot [35], une extension conservative de la bibliothèque standard des réels de Coq [148] présentée en section 2.2.1.

Ce chapitre est voué à la formalisation des espaces dits préhilbertiens et hilbertiens. La majeure partie des notions géométriques palpables, comme les concepts de plan, de droite ou d'angle, forment la géométrie dite euclidienne. Ils ont pour cadre les espaces euclidiens, *i.e.* des espaces vectoriels de dimension finie munis d'un produit scalaire, à partir duquel il est possible d'inférer une norme. Dans l'objectif d'adapter les résultats d'algèbre linéaire des espaces euclidiens aux espaces fonctionnels, les espaces dits préhilbertiens ont commencé à être étudiés dès la fin du XIX^{e} siècle [153]. Les espaces préhilbertiens sont une généralisation des espaces euclidiens lorsque la dimension est potentiellement infinie. La terminologie d'espaces de Hilbert (ou espaces hilbertiens) est quant à elle réservée aux espaces préhilbertiens complets (voir section 7.3).

En section 7.1, nous présentons la bibliothèque Coquelicot sur laquelle se base notre formalisation : en particulier, nous décrivons la manière dont les concepts topologiques y sont définis. La section 7.2 introduit quelques définitions complémentaires concernant les sous-espaces. Nous décrivons ensuite, dans la section 7.3, la manière dont nous avons défini les espaces de Hilbert en Coq. Dans la section 7.4, nous exhibons quelques résultats fondamentaux d'analyse des espaces de Hilbert qui ont été formalisés. Enfin, dans la section 7.5, nous nous focalisons sur des espaces de fonctions et plus particulièrement sur l'espace des fonctions linéaires continues.

Remarque. *La grande majorité des développements mathématiques relevant de l'analyse fonctionnelle (e.g. l'ouvrage fondateur de Brézis [42]) font appel a minima à des variantes faibles de l'axiome du choix [110]. Certains résultats présentés dans ce chapitre ainsi que dans le chapitre 8 sont démontrés en logique classique (pour rappel, les résultats classiques sont étiquetés par le symbole* $\text{🐓}_{P\vee\neg P}$*). Il est néanmoins possible de se restreindre à des hypothèses de décidabilité bien précises : certaines de ces hypothèses donnent lieu à des discussions, réunies dans la section 8.6 à la fin du chapitre 8. Nous utilisons en outre l'axiome* `ProofIrrelevance`*, qui permet de gérer l'égalité entre éléments de types dépendants, comme en section 7.5.3 pour le type* `clm`*. Enfin, nous utilisons l'axiome* `FunctionalExtensionality` *afin de réduire l'égalité de fonctions à une égalité point à point, principe généralement admis dans les preuves mathématiques.*

7.1 Prérequis sur la bibliothèque Coquelicot

Les travaux présentés dans le présent chapitre ainsi que dans le chapitre 8 reposent essentiellement sur Coquelicot, une extension conservative de la bibliothèque standard des nombres réels conçue pour pallier les défauts de cette dernière et destinée à la formalisation de résultats d'analyse [35, 134]. La particularité de Coquelicot est l'utilisation du formalisme des filtres pour la définition des concepts topologiques (voir section 7.1.1). Ce formalisme étant très général, il a été possible de généraliser les notions mathématiques à des structures algébriques plus générales que $\mathbb{R}$, comme les anneaux ou les espaces normés complets (voir section 7.1.2). La bibliothèque Coquelicot comporte l'essentiel des notions permettant de démontrer des résultats concernant la topologie, les limites, les séries entières, la dérivation ou encore l'intégration, la plupart de ces concepts étant définis comme des fonctions totales. Notons également que Coquelicot est basée sur l'axiomatisation des réels de Coq mais n'utilise pas l'axiome du tiers-exclu.

7.1.1 Topologie générales, filtres et limites

La topologie générale permet d'unifier les notions de voisinage et de limite. Au cours de la première moitié du XX^{e} siècle, plusieurs formalismes ont été développés pour définir les limites dans des espaces topologiques généraux. Les deux approches les plus courantes sont les suites généralisées (*nets* en anglais) remontant aux travaux de Moore et Smith en 1922 [152] et les filtres, dont la première mention est attribuée à Cartan en 1937 [48]. En HOL-Light, Harrison utilise les suites généralisées pour généraliser les notions de convergence [103]. La bibliothèque Coquelicot utilise quant à elle le formalisme des filtres [35, 133], dans lequel il est possible d'exprimer des notions usuellement exprimées avec des suites, comme la convergence, le critère de Cauchy, la continuité d'une fonction ou la complétude et la fermeture d'un espace. Un filtre est un ensemble de voisinages, plus précisément une collection `F: (E -> Prop) -> Prop` de sous-ensembles [2] d'un ensemble `E : Type` vérifiant les propriétés de la classe `Filter` définie ci-dessous (la figure 7.1 présente quelques exemples de filtres).

```
Class Filter {E : Type} (F : (E → Prop) → Prop) := {
  filter_true : F (fun _ ⇒ True) ;
  filter_and : forall P Q : E → Prop, F P → F Q
                          → F (fun x ⇒ P x ∧ Q x) ;
  filter_imp : forall P Q : E → Prop,
                   (forall x, P x → Q x) → F P → F Q }.
```

Il est également nécessaire d'introduire un concept fréquemment utilisé, à savoir la notion de filtre propre :

```
Class ProperFilter {E : Type} (F : (E → Prop) → Prop) := {
  filter_ex : forall P, F P → exists x, P x ;
  filter_filter :> Filter F }.
```

2. Pour `D : E -> Prop`, `D x` signifie que `x` appartient au sous-espace `D` (voir section 7.2).

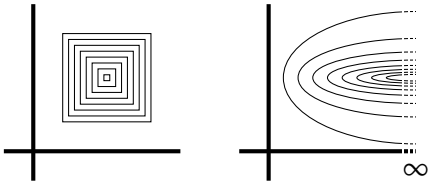


FIGURE 7.1 – Filtres convergents : vers un point fini (gauche), vers l'infini (droite)

Hölzl, Immler et Huffmann utilisent les filtres comme clefs de voûte des notions topologiques dans une formalisation de l'analyse mathématique en Isabelle/HOL [109]. Coquelicot s'inspire de ces travaux pour définir les notions de convergence, même les plus élémentaires [133]. Les filtres `eventually`, `locally` et `Rbar_locally` (décrits ci-dessous) permettent de définir la convergence des suites réelles et des fonctions.

Le filtre `eventually` contient l'ensemble des parties de `nat` qui contiennent tous les entiers naturels à partir d'un certain rang.

```
Definition eventually (P : nat → Prop) :=
            exists N : nat, forall n, N <= n → P n.
```

Pour `x : T` fixé, le filtre `locally x` contient tous les voisinages du point `x` (voir figure 7.1). Le filtre `Rbar_locally x` est analogue à `locally x` pour `x : Rbar`. Dans Coquelicot, un élément de `Rbar` est soit construit à partir d'un élément de type `R` (via la constructeur `Finite` qui plonge les réels dans `Rbar`), soit égal à $\pm\infty$.

```
Definition locally (x : T) (P : T → Prop) :=
            exists eps : posreal, forall y, ball x eps y → P y.

Definition Rbar_locally (x : Rbar) (P : R → Prop) := match x with
    | Finite a ⇒ locally a P
    | p_infty ⇒ exists M : R, forall t, M < t → P t
    | m_infty ⇒ exists M : R, forall t, t < M → P t
  end.
```

La prédicat `ball` apparaissant dans le code Coq ci-dessus est une généralisation de la notion de boule pour des espaces qui ne sont pas nécessairement munis de métrique. Pour donner une intuition, `ball x eps y` signifie qu'il y a un « écart » inférieur à `eps` entre les points `x` et `y` (ou alternativement que `y` appartient à une boule généralisée de centre `x` et de rayon `eps`). Cela permet de dire, pour `eps` suffisamment petit, que `x` et `y` sont proches sans utiliser la notion de distance. L'objet `ball` est formellement défini en section 7.1.2.

Pour définir les notions de convergence, il faut d'abord introduire un constructeur `filtermap` qui permet de construire l'image d'un filtre par une fonction puis un prédicat `filterlim` qui pourra être instancié comme la limite d'une suite ou d'une fonction :

```
Definition filtermap {T U : Type} (f : T → U)
          (F : (T → Prop) → Prop) := fun P ⇒ F (fun x ⇒ P (f x)).
```

```
Definition filterlim {T U : Type} (f : T → U) F G :=
                    forall P : T → Prop, (filtermap f F) P → G P.
```

Ainsi est-il possible d'exprimer la relation `is_lim_seq` caractérisant la limite (à valeurs dans $\overline{\mathbb{R}}$) d'une suite réelle :

```
Definition is_lim_seq (u : nat → R) (l : Rbar) :=
              filterlim u eventually (Rbar_locally l).
```

Cela signifie que la suite `u` reste dans le voisinage du point `l` à partir d'un certain rang.

Sur le même modèle que pour la convergence des suites, Coquelicot comporte un prédicat `is_lim` pour la convergence d'une fonction de type `R -> R` en un point `x` :

```
Definition is_lim (f : R → R) (x l : Rbar) :=
    filterlim f (Rbar_locally' x) (Rbar_locally l).
```

La propriété `is_lim f x l` signifie que l'image réciproque par la fonction `f` de tout voisinage de la limite `l` est un voisinage du point `x`. Il y a en fait une légère subtilité, liée à l'occurrence d'un filtre `Rbar_locally' x` qui est en fait analogue à `Rbar_locally x` mais permet d'exclure le point `x` des voisinages appartenant au filtre. Cela permet de traiter le cas où le point `x` n'est pas contenu dans l'image réciproque du voisinage de `l`.

7.1.2 Hiérarchie algébrique de Coquelicot

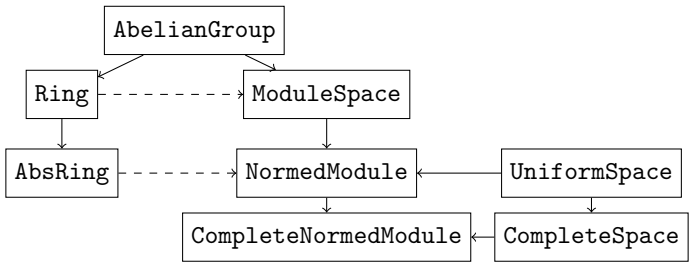


FIGURE 7.2 – Hiérarchie algébrique de Coquelicot. Les flèches pleines signifient que la structure est construite à partir d'une autre structure. Les flèches en pointillés signifient que la structure est paramétrée par une autre structure (*e.g.* un module est paramétré par un anneau sous-jacent).

Coquelicot n'est pas seulement une bibliothèque d'analyse réelle. Elle permet en effet de démontrer des résultats valables dans des structures algébriques plus générales. La définition des classes d'espaces a été réalisée à l'aide des structures canoniques [143], un mécanisme permettant de construire une hiérarchie de types et de surcharger les notations. La hiérarchie algébrique de Coquelicot (voir figure 7.2), inspirée des travaux antérieurs de Hölzl, Immler et Huffman [109], comporte des structures de natures

différentes. À la base, nous trouvons les structures purement algébriques que sont les groupes abéliens (`AbelianGroup`), les anneaux (`Ring`) et les modules (`ModuleSpace`). Ces derniers, qui sont au coeur des développements présentés dans ce chapitre et dans le chapitre 8, peuvent être vus comme des espaces vectoriels « appauvris » dont l'ensemble scalaire n'est pas un corps mais un anneau. La structure de `ModuleSpace` est définie comme suit (`Mixin` est le constructeur usuel des structures canoniques) :

```
Record mixin_of (K : Ring) (V : AbelianGroup) := Mixin {
  scal : K → V → V ;
  ax1 : forall x y u, scal x (scal y u) = scal (mult x y) u ;
  ax2 : forall u, scal one u = u ;
  ax3 : forall x u v, scal x (plus u v) = plus (scal x u) (scal x v) ;
  ax4 : forall x y u, scal (plus x y) u = plus (scal x u) (scal y u)
}.
```

En parallèle et afin d'introduire les notions topologiques de base, la hiérarchie de Coquelicot a été enrichie d'une structure d'espaces dits uniformes (`UniformSpace`). Ces espaces, dont la première mention remonte à Weil à la fin des années 1930 [204, 41], généralisent la notion d'espaces métriques. La structure `UniformSpace` est formalisée à partir d'un objet `ball` (boule généralisée)[3] qui permet de définir une notion de voisinage, vérifiant les propriétés de réflexivité, de symétrie et de transitivité :

```
Record mixin_of (M : Type) := Mixin {
  ball : M → R → M → Prop ;
  ax1 : forall x (e : posreal), ball x e x ;
  ax2 : forall x y e, ball x e y → ball y e x ;
  ax3 : forall x y z e1 e2, ball x e1 y → ball y e2 z
                              → ball x (e1 + e2) z
}.
```

En revanche, la propriété de distinguabilité des espaces métriques, *i.e.* `ball x 0 y <-> x = y` n'est pas requise *a priori*. Il a toutefois été nécessaire d'exprimer un concept pour les points arbitrairement proches par la relation

```
close x y := forall eps: posreal, ball x eps y.
```

Historiquement, la notion d'espace uniforme a été définie de façon à avoir une structure minimale pour définir les notions de continuité et de suites de Cauchy ; cela est également valable pour les filtres [41]. La notion de continuité d'une fonction de type `T -> U` avec `T, U : UniformSpace` en un point `x : T` a été formalisée :

```
Definition continuous {T U : UniformSpace} (f : T → U) (x : T) :=
                        filterlim f (locally x) (locally (f x)).
```

3. Dans les énoncés mathématiques, la propriété `ball x e y` est noté $y \in \mathcal{B}(x, e)$.

Cette définition généralise la définition « usuelle » de continuité, *i.e.* f est continue en un point x si et seulement si la limite de f en x est égale à $f(x)$, ce qui est équivalent à la propriété `is_lim f x (f x)`.

Un filtre est dit de Cauchy s'il contient des boules généralisées arbitrairement petites (cette définition généralise la notion de suite de Cauchy).

```
Definition cauchy {T : UniformSpace} (F : (T → Prop) → Prop) :=
             forall eps : posreal, exists x, F (ball x eps).
```

Pour un espace uniforme `T`, il est possible d'exprimer la notion de fermeture d'un sous-espace de `T` défini par son prédicat caractéristique `D : T -> Prop`.

```
Definition closed {T : UniformSpace} (D : T → Prop) :=
     forall x, not (locally x (fun x : T ⇒ not (D x))) → D x.
```

Cette définition signifie[4] que le complémentaire `C` de `D` est un ouvert (`open` en Coq) de `T`, *i.e.* pour tout élément `x` de `C`, il existe une boule de centre `x` et contenue dans `C`.

À partir de la structure `UniformSpace`, la structure `CompleteSpace` des espaces complets a été formalisée. Un espace complet est un espace uniforme `T` enrichi d'une fonction totale limite `lim : ((T → Prop) → Prop) → T` qui à tout filtre associe une limite dans `T`, cette limite vérifiant deux propriétés supplémentaires :

```
Record mixin_of (T : UniformSpace) := Mixin {
  lim : ((T → Prop) → Prop) → T ;
  ax1 : forall F, ProperFilter F → cauchy F →
         forall eps : posreal, F (ball (lim F) eps) ;
  ax2 : forall F1 F2, filter_le F1 F2 → filter_le F2 F1 →
                close (lim F1) (lim F2)
}.
```

Le premier axiome `ax1` exprime la convergence de tout filtre propre et de Cauchy sur `T` ; en d'autres termes : tout filtre propre contenant des boules arbitrairement petites contient des boules arbitrairement petites dont le centre est la limite du filtre. Le second axiome exprime le fait que les limites de deux filtres « équivalents » soient arbitrairement proches.

À partir des deux structures indépendantes `ModuleSpace` et `UniformSpace`, il est possible de définir la structure de module normée `NormedModule`, *i.e.* un `ModuleSpace` muni d'une norme, à partir de laquelle il est possible de définir une pseudométrique `ball` et donc d'inférer un `UniformSpace`. À partir d'un `CompleteSpace` et d'un `NormedModule`, la structure de `CompleteNormedModule` est obtenue.

Les structures canoniques permettent ensuite de construire des instances de ces structures algébriques pour des espaces usuels [143]. Par exemple, dans Coquelicot, `R` a été muni d'une structure d'anneau `Ring`. Cela revient à donner les lois d'anneau (ici

4. En fait, depuis la version 3 de Coquelicot, `closed` est défini comme la contraposée du prédicat « le complémentaire est ouvert », et n'est donc plus équivalent à `open (fun x => not (D x))` en logique intuitionniste. L'usage du tiers-exclu permet cependant de retrouver l'équivalence.

`Rplus` et `Rmult`) puis de prouver qu'elles satisfont les propriétés d'un anneau. En fait, `R` a été muni de structures algébriques plus complexes, comme la structure de `NormedModule`, de `CompleteSpace`, puis enfin de `CompleteNormedModule` [134, 35]. C'est également le cas de l'ensemble `C` des nombres complexes.

7.2 À propos des sous-espaces

Certaines propriétés topologiques standards, comme le caractère ouvert ou fermé d'un ensemble, n'ont de sens que pour un sous-espace relativement à un espace plus grand (nous utiliserons respectivement les termes sous-espace et sur-espace). Or, en Coq, contrairement à d'autres assistants de preuves comme PVS [160], la définition de sous-structures et l'obtention de relations de sous-typages sont connues comme des problèmes difficiles [140, 7]. Il existe plusieurs manières de pallier cette difficulté.

7.2.1 Définitions et compatibilité des sous-espaces

Une première solution est l'utilisation de types dépendants consistant à définir un type enregistrement (`Record`) dont le premier champ est un élément du sur-espace et le second champ une preuve d'appartenance au sous-espace. C'est parfois l'approche adoptée dans ce manuscrit, *e.g.* pour définir les formes linéaires continues (voir section 7.5.3). Cependant, cela nécessite de redéfinir les opérations et relations du sur-espace pour le sous-espace [5]. Par exemple, si nous définissons le type des nombres réels supérieurs ou égaux à 7 (que nous notons `R_le7` pour l'exemple), il est nécessaire de redéfinir une relation d'ordre `Rlt' : R_le7 -> R_le7 -> Prop` bien que celle-ci coïncide avec la relation d'ordre `Rlt` pour les nombres réels.

La seconde solution est la caractérisation d'un sous-espace d'un espace `E` par un prédicat de type `E -> Prop`. Ce prédicat peut également être vu comme la fonction caractéristique du sous-espace. Cette approche a été intensivement utilisée par Gonthier *et al.* dans la preuve formelle du théorème de Feit-Thompson [86], un résultat fondamental en théorie des groupes finis [6]. Nous adoptons très fréquemment cette solution dans la suite du document car elle présente l'avantage de pouvoir dire d'un même élément (au sens le plus fort, c'est-à-dire d'un même terme Coq) `x` qu'il appartient à `E` (par typage) et qu'il est un élément du sous-espace `phi` (*i.e.* `phi x`). Dans les définitions et énoncés mathématiques, nous utiliserons à la fois les notations $x \in \varphi$ et $\varphi(x)$ pour dénoter l'appartenance d'un élément $x : E$ à un sous-espace $\varphi : E \to Prop$.

Pour E : `AbelianGroup` et $\varphi : E \to Prop$ le prédicat d'un sous-espace de E, il peut s'avérer nécessaire de supposer ou de vérifier que φ est un sous-groupe de E. Cela signifie que $0_E \in \varphi$, et que φ est stable par l'addition et l'opposé. La conjonction de ces trois propriétés est équivalente à la propriété suivante :

5. L'utilisation du mécanisme des coercions permet cependant de surcharger les notations.

6. Cette approche s'avère d'autant plus adéquate que l'appartenance d'un élément à un sous-groupe fini est décidable et donc que le prédicat caractéristique associé est décidable.

```
Definition compatible_g {E : AbelianGroup} (phi : E → Prop) :=
  (forall x y, phi x → phi y → phi (plus x (opp y))) ∧ (exists x, phi x).
```

Nous utiliserons la terminologie `AbelianGroup`-compatible pour caractériser les prédicats vérifiant cette propriété. De même, il est possible de définir les sous-espaces `ModuleSpace`-compatibles d'un module donné, par la propriété :

```
Definition compatible_m {E : ModuleSpace} (phi : E → Prop) :=
  compatible_g phi ∧ (forall (x : E) (k : R), phi x → phi (scal k x)).
```

7.2.2 Topologie et sous-espaces

Comme mentionné en section 7.1 puis au début de la section courante, certaines notions topologiques présentes dans la bibliothèque Coquelicot n'ont de sens que pour des sous-espaces (*e.g.* le caractère ouvert ou fermé). Une difficulté supplémentaire apparaît lorsqu'il devient nécessaire de transposer aux sous-espaces une propriété qui leur soit propre, *i.e.* indépendante du sur-espace à partir duquel ils sont définis. Considérons par exemple un espace E : `UniformSpace`. Nous pouvons prouver la propriété de complétude de cet espace, puis le munir de la structure de `CompleteSpace`. Considérons maintenant un sous-espace φ de type $E \to Prop$ et imaginons qu'il faille prouver la complétude de φ. La propriété de complétude définie en section 7.1.2 ne peut être énoncée qu'à partir d'un `UniformSpace`, or φ ne peut pas être muni d'une telle structure puisqu'il s'agit d'un prédicat caractéristique de type $E \to Prop$. Il est donc nécessaire de définir une notion de complétude qui soit propre aux sous-espaces.

Pour des raisons de typage, nous ne pouvons pas définir de filtre sur un sous-espace $\varphi : E \to Prop$. En revanche, nous pouvons dire d'un filtre sur E (de type $(E \to Prop) \to Prop$) qu'il est *induit* sur φ, *i.e.* qu'il ne contient que des parties de E ayant une intersection non-vide avec φ :

Définition 7.1 (Filtre induit). *Soit E : `Type`, $\mathscr{F} : (E \to Prop) \to Prop$ un filtre sur E. Soit $\varphi : E \to Prop$ un sous-espace de E. $\mathscr{F}$ est induit sur φ ssi :*

$$\forall \psi : E \to Prop, \mathscr{F}(\psi) \Rightarrow \exists x \in E, \psi(x) \wedge \varphi(x).$$

Le terme Coq associé à la notion de filtre induit est noté `induced F phi`.

Pour définir la propriété de complétude des sous-espaces, nous nous inspirons de la notion de complétude des espaces dans la bibliothèque Coquelicot, utilisée pour définir la structure `CompleteSpace` (voir section 7.1.2).

Commençons par la définition la plus générale, définissant la complétude des sous-espaces d'un `UniformSpace` (qui n'est pas nécessairement lui-même complet) :

```
Definition complete_subset (phi : E → Prop) :=
  exists (lime : ((E → Prop) → Prop) → E),
    (forall F, ProperFilter F → cauchy F → induced F phi →
      phi (lime F) ∧ forall eps : posreal, F (ball (lime F) eps)).
```

À des fins de simplification, lorsque E : `CompleteSpace`, nous souhaitons utiliser une version forte de la complétude, qui impose que la fonction limite `lime` de la définition `complete_subset` soit la fonction limite `lim` associée au sur-espace. Cette version forte de la complétude des sous-espaces est notée `my_complete` et définie par :

```
Definition my_complete (phi : E → Prop) :=
  forall (F : (E → Prop) → Prop),
    ProperFilter F → cauchy F → induced F phi → phi (lim F).
```

Nous démontrons assez facilement l'implication suivante en instanciant `lime` avec la fonction `lim` :

```
Context {E : CompleteSpace} {phi : E → Prop}

Lemma my_complete_complete_subset : my_complete phi → complete_subset phi.
Proof. (* ... *) Qed.
```

Dans la suite du document, lorsque nous faisons référence à un sous-espace φ complet, cela signifie `complete_subset phi` lorsque le sur-espace n'est pas complet (`UniformSpace`, `NormedModule`, `PreHilbert`, etc) et cela signifie `my_complete` lorsque le sur-espace est complet (`CompleteSpace`, `CompleteNormedModule`, `Hilbert`).

Notons également que lorsque E : `CompleteSpace` et φ est fermé dans E (voir section 7.1.2), nous pouvons prouver que φ est complet (au sens `my_complete phi` puis au sens `complete_subset phi`). En résumé, nous avons les implications suivantes entre fermeture et complétude des sous-espaces.

```
Context {E : CompleteSpace} {phi : E → Prop}

Lemma closed_my_complete : closed phi → my_complete phi.
Proof. (* ... *) Qed.

Lemma closed_complete_subset : closed phi → complete_subset phi.
Proof. (* ... *) Qed.
```

En revanche, l'équivalence `my_complete phi <-> closed phi` n'est pas vraie en général, mais l'est pour E : `CompleteNormedModule`.

7.3 Formalisation des espaces de Hilbert

Nous commençons tout d'abord par formaliser les structures d'espace préhilbertien et d'espace de Hilbert. Nous étendons pour cela la hiérarchie algébrique de la bibliothèque Coquelicot présentée en section 7.1.2.

Un espace préhilbertien réel (`PreHilbert`) est un module dont l'anneau scalaire est l'ensemble des nombres réels (`R_Ring` étant la structure d'anneau sur `R`), muni d'un produit scalaire `inner`. Celui-ci doit par ailleurs vérifier les propriétés usuelles d'un produit scalaire. La structure canonique correspondante est la suivante :

```
Record mixin_of (E : ModuleSpace R_Ring) := Mixin {
  inner : E → E → R ;
  ax1 : forall (x y : E), inner x y = inner y x ;
  ax2 : forall (x : E), 0 <= inner x x ;
  ax3 : forall (x : E), inner x x = 0 → x = zero ;
  ax4 : forall (x y : E) (k : R),
             inner (scal k x) y = k * inner x y ;
  ax5 : forall (x y z : E),
             inner (plus x y) z = inner x z + inner y z }.
```

Dans les énoncés mathématiques, le produit scalaire `inner` est noté $\langle \cdot, \cdot \rangle_E$. À partir du produit scalaire, nous sommes en mesure de définir une norme et de prouver que tout espace préhilbertien est un `NormedModule`[7]. Un espace de Hilbert est un espace préhilbertien complet, *i.e.* muni d'une limite vérifiant les propriétés de complétude au sens des filtres (voir la définition de `CompleteSpace` en section 7.1.2).

```
Record mixin_of (E : PreHilbert) := Mixin {
  lim : ((E → Prop) → Prop) → E ;
  ax1 : forall F, ProperFilter F → cauchy F →
        forall eps : posreal, F (ball (lim F) eps) ;
  ax2 : forall F1 F2, filter_le F1 F2 → filter_le F2 F1 →
              close (lim F1) (lim F2) }.
```

Nous prouvons qu'un espace muni de la structure `Hilbert` est également muni de la structure `PreHilbert` et de la structure `CompleteNormedModule`.

L'extension de la hiérarchie algébrique est synthétisée par la figure 7.3.

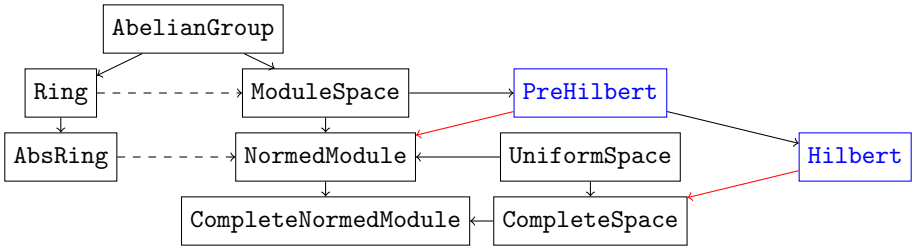


FIGURE 7.3 – Hiérarchie étendue aux espaces `PreHilbert` et `Hilbert`. Les flèches pleines noires signifient que la structure est construite à partir d'une autre structure. Les flèches en pointillés signifient que la structure est paramétrée par une autre structure (*e.g.* un module est paramétré par un anneau sous-jacent). Les flèches rouges signifient que nous avons prouvé qu'une structure a également les propriétés d'une autre structure (*e.g.* un espace de Hilbert est complet).

7. De surcroît, la norme $\|\cdot\|$ dérivée du produit scalaire vérifie la propriété $\|x\| = 0 \Leftrightarrow x = 0$, ce qui n'est pas le cas de toutes les normes.

7.4 Propriétés géométriques et espaces de Hilbert

Dans cette section, nous définissons les concepts-clefs d'analyse des espaces de Hilbert, comme le projeté orthogonal d'un vecteur et le complément orthogonal d'un sous-espace. Nous prouvons par ailleurs quelques résultats importants concernant ces objets géométriques.

7.4.1 Identité du parallélogramme, inégalité de Cauchy-Schwarz

Nous commençons par prouver formellement deux lemmes simples concernant le produit scalaire, à savoir l'identité du parallélogramme et l'inégalité de Cauchy-Schwarz :

Lemme 7.1. Identité du parallélogramme
Soit E : `PreHilbert`. *Alors :*

$$\forall x, y \in E, \langle x+y, x+y\rangle_E + \langle x-y, x-y\rangle_E = 2\left(\langle x, x\rangle_E + \langle y, y\rangle_E\right).$$

Lemme 7.2. Inégalité de Cauchy-Schwarz
Soit E : `PreHilbert`. *Alors :*

$$\forall x, y \in E, \sqrt{\langle x, y\rangle_E} \leqslant \langle x, x\rangle_E \cdot \langle y, y\rangle_E.$$

Assez simple, la démonstration de ces deux résultats repose sur les propriétés du produit scalaire, notamment sur sa bilinéarité [50, lemmes 173 et 174]. Ces résultats sont néanmoins indispensables aux démonstrations d'autres théorèmes. En effet, le lemme 7.1 (inégalité du parallélogramme) est nécessaire à la démonstration du théorème 7.3 présentée succinctement ci-dessous. Une variante de l'inégalité du parallélogramme est également nécessaire à la preuve du lemme 7.4. Quant au lemme 7.2, il permet de prouver la continuité du projeté orthogonal (en tant que fonction prenant un vecteur en entrée et renvoyant son projeté orthogonal) ainsi que le théorème 8.2 (théorème de Riesz–Fréchet).

7.4.2 Projeté orthogonal

Si la notion de projeté orthogonal d'un vecteur sur un sous-espace paraît intuitive (voir figure 7.4), la preuve de son existence et de son unicité est loin d'être triviale.

Théorème 7.3. Projeté orthogonal, sous-espace complet $_{P\vee\neg P}$
Soit E : `PreHilbert`. *Soit* $\varphi : E \to$ `Prop`, φ *non vide, complet et* `ModuleSpace`*-compatible*[8]. *Alors, pour tout* $u \in E$, *il existe un unique vecteur* $P_\varphi(u) \in E$ *tel que :*

$$\varphi(P_\varphi(u)) \quad \wedge \quad \|u - P_\varphi(u)\|_E = \inf_{w \in E \wedge \varphi(w)} \|u - w\|_E.$$

8. En réalité, nous prouvons formellement que le résultat reste valable pour un sous-espace non vide, complet et convexe : nous démontrons que tout sous-espace `ModuleSpace`-compatible est convexe.

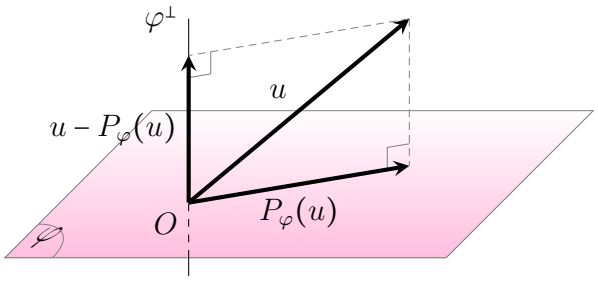


FIGURE 7.4 – Projetés orthogonaux sur un sous-espace φ et sur son complément orthogonal $\varphi^\perp$

$P_\varphi(u)$ est appelé *projeté orthogonal* de u sur φ (le concept de projeté orthogonal est illustré par la figure 7.4).

Pour prouver formellement ce résultat, nous nous basons sur la démonstration papier de Clément et Martin [50, théorème 186], qui n'utilise pas la notion de filtre. Comme nous utilisons la bibliothèque Coquelicot, dans laquelle les notions de limites utilisent les filtres, nous devons dans un premier temps interpréter la démonstration dans ce formalisme. Posons $\delta = \inf_{w \in E \wedge \varphi(w)} \|u - w\|_E$. Le projeté orthogonal est construit comme la limite du filtre :

```
F := fun (V:E→ Prop) ⇒ exists eps:posreal, forall x,
          phi x →norm (minus u x) < delta + eps →V x.
```

Pour donner du sens à cette limite, nous prouvons tout d'abord que le filtre `F` est propre et de Cauchy. Ensuite, nous utilisons la complétude du sous-espace φ pour montrer l'appartenance de la limite au sous-espace. La démonstration est par ailleurs classique à cause de problèmes de décidabilité liés à la borne inférieure dans la définition de δ (voir détails en section 8.6).

Ensuite, nous formalisons le lemme de caractérisation suivant, qui généralise la notion de projeté orthogonal à tout sous-espace `ModuleSpace`-compatible de E :

Lemme 7.4. Caractérisation du projeté orthogonal  $_{P \vee \neg P}$

Soit E : `PreHilbert`. Soit $\varphi : E \to$ `Prop`, φ `ModuleSpace`-compatible. Soit $u, v \in E$. Supposons que $v \in \varphi$. Alors :

$$\left(\|u - v\|_E = \inf_{w \in E \wedge \varphi(w)} \|u - w\|_E \right) \iff (\forall w \in E,\ \varphi(w) \implies \langle v, w \rangle_E = \langle u, w \rangle_E) .$$

Ce lemme, valable pour n'importe quel sous-espace φ `ModuleSpace`-compatible, est un lemme de caractérisation et n'assure pas l'existence du projeté orthogonal, qui n'est prouvable que si φ est complet (d'après le théorème 7.3).

La démonstration est assez fastidieuse et repose sur la manipulation d'égalités et d'inégalités impliquant des produits scalaires. La preuve de ces équations utilise à plusieurs reprises une variante simplifiée de l'inégalité du parallélogramme (lemme 7.1).

Ces caractérisations ne permettent pas d'obtenir constructivement une fonction ayant pour paramètres un vecteur u et un sous-espace φ et renvoyant le projeté orthogonal $P_\varphi(u)$. Une solution simple serait d'utiliser l'opérateur `iota` de Hilbert disponible dans la bibliothèque standard de Coq, qui est une version faible de l'opérateur `epsilon` de Hilbert dans le cas où un argument d'unicité peut être fourni. Néanmoins, ces opérateurs nécessitent l'axiome `Epsilon` de Hilbert.

Il est cependant possible, dans le cas particulier où E : `Hilbert`, de n'utiliser ni le tiers-exclu ni l'axiome `Epsilon` de Hilbert. En effet, lorsque E : `CompleteSpace` (et *a fortiori* lorsque E : `Hilbert`), la bibliothèque Coquelicot fournit un opérateur `iota` ne nécessitant aucun axiome particulier, sinon l'axiomatisation des réels de la bibliothèque standard [35, 134]. Cet opérateur possède un comportement analogue à l'opérateur `iota` de la bibliothèque standard et est construit comme la limite d'un filtre bien choisi. Dans ce manuscrit, la notation `iota` fera toujours référence à l'opérateur de Coquelicot. Cet opérateur nous permet de construire la fonction `proj` :

```
Definition proj {E : Hilbert} (phi : E → Prop) :=
  fun u : E ⇒ iota (fun v : E ⇒ phi v ∧ norm (minus u v)
    = Glb_Rbar (fun r ⇒ exists w:E, phi w ∧ r = norm (minus u w))).
```

Dans la définition Coq ci-dessus, `Glb_Rbar P` est la construction de Coquelicot correspondant à la plus grande borne inférieure du sous-espace `P : Rbar -> Prop`. Dans les énoncés mathématiques, la fonction totale de projection orthogonale est notée P_φ.

De façon surprenante, cette fonction explicite n'est utilisée ni dans la preuve du théorème de Lax–Milgram (théorème 8.4), ni dans la preuve des résultats intermédiaires. Les résultats de caractérisation présentés ci-dessus, *i.e.* le théorème 7.3 et le lemme 7.4 s'avèrent en effet suffisants. En revanche, pour démontrer la complétude des sous-espaces de dimension finie des espaces de Hilbert (théorème 8.9), il faut construire l'image d'un filtre par la fonction projeté orthogonal. Il devient donc nécessaire d'utiliser l'objet `proj` en tant que fonction explicite, puis de démontrer qu'elle possède certaines bonnes propriétés. La première propriété, simple à démontrer, est la linéarité. La seconde propriété est le caractère contractant du projeté orthogonal :

```
forall x y, norm (proj phi x - proj phi y) <= norm (x - y).
```

7.4.3 Complément orthogonal et somme directe

Définition 7.2 (Complément orthogonal).
Soit E : `PreHilbert`. Soit $\varphi : E \to$ `Prop`, φ `ModuleSpace`-compatible. Son complément orthogonal est le sous-ensemble $\varphi^\perp$ de E défini de la manière suivante :

```
Definition orth_compl := fun x : E ⇒ forall y, phi y → inner x y = 0.
```

Lorsque φ est un sous-espace complet, tout vecteur $u \in E$ peut être décomposé comme la somme de deux vecteurs respectivement à valeurs dans φ et dans $\varphi^\perp$. Nous prouvons que la décomposition sur φ et $\varphi^\perp$ est unique (il s'agit d'une conséquence

de l'unicité de la projection orthogonale assurée par le théorème 7.3). Cela permet d'introduire la notion de somme directe :

Définition 7.3 (Somme directe). *Soit* $E : ModuleSpace$, $\varphi, \psi, \pi : E \to Prop$. *On dit que* $\varphi = \psi \oplus \pi$ *(*φ *est la somme directe de* ψ *et* π*) ssi :*

$$\forall u \in E, \varphi(u) \Rightarrow \exists! a, b \in E, a \in \psi \wedge b \in \pi \wedge u = a + b$$

Nous avons donc $E = \varphi \oplus \varphi^\perp$ (notation pour $\lambda x.\texttt{True} = \varphi \oplus \varphi^\perp$). Par ailleurs, cette décomposition unique peut être donnée explicitement : $u = P_\varphi(u) + (u - P_\varphi(u))$ (voir figure 7.4). Nous proposons plusieurs formalisations de la propriété « φ et φ' sont la somme directe d'un ensemble E », prouvées équivalentes par une série d'implications. En pratique, la seule des formalisations équivalentes qui est utilisée est la suivante :

```
Definition direct_sumable phi phi' := forall x:E, phi x → phi' x → x = 0.
```

Nous prouvons enfin un dernier lemme de caractérisation utilisant le fait qu'un sous-module d'un espace complet est en somme directe avec son complément orthogonal.

Lemme 7.5. Caractérisation des sous-espaces orthogonaux  $_{P \vee \neg P}$
Soit E : *PreHilbert. Soit* $\varphi : E \to$ *Prop,* φ *ModuleSpace-compatible et complet. Soit* $u, v \in E$. *Supposons que* $\varphi(v)$. *Alors :*

$$\left(\|u - v\|_E = \inf_{w \in E \wedge \varphi(w)} \|u - w\|_E \right) \implies (\varphi(u) \iff v = u) \wedge (\varphi^\perp(u) \iff v = 0)\,.$$

Ce lemme est utilisé dans la preuve du théorème 8.2 (Riesz–Fréchet) et est donc nécessaire à la preuve du théorème 8.4 (Lax–Milgram). Un corollaire assure que si $x \in \varphi$, $P_\varphi(x) = x$ et pour $x \in \varphi^\perp$, $P_\varphi(x) = 0$.

7.5 Fonctions linéaires continues

Le théorème de Lax–Milgram s'applique à des fonctions linéaires continues des espaces de Hilbert. La notion de linéarité, très simple à caractériser, est présentée en section 7.5.1. En section 7.5.2, nous définissons la notion de norme subordonnée d'une fonction linéaire. Nous prouvons qu'il existe plusieurs définitions équivalentes de la notion de continuité pour les fonctions linéaires puis choisissons de définir l'espace des fonctions linéaires continues à l'aide de la norme subordonnée en section 7.5.3.

7.5.1 Fonctions linéaires et bilinéaires

Soit E, F : `ModuleSpace`, nous définissons un prédicat `is_linear_mapping` sur $E \to F$ caractérisant les fonctions linéaires :

```
Definition is_linear_mapping {E F : ModuleSpace} (f : E → F) :=
      (forall (x y : E), f (plus x y) = plus (f x) (f y))
    ∧ (forall (x : E) (k : R), f (scal k x) = scal k (f x)).
```

Nous prouvons par ailleurs que le sous-ensemble `is_linear_mapping` de $E \to F$ est `AbelianGroup`-compatible.

Ensuite, nous définissons un prédicat similaire pour les applications bilinéaires (*i.e.* les fonctions de deux arguments qui sont linéaires sur chacun de leurs arguments), et prouvons qu'il est également `AbelianGroup`-compatible :

```
Definition is_bilinear_mapping
           {E F G : ModuleSpace} (f : E → F → G) :=
 (forall (x : E) (y : F) (k : R), f (scal k x) y = scal k (f x y)) ∧
 (forall (x : E) (y : F) (k : R), f x (scal k y) = scal k (f x y)) ∧
 (forall (x y : E) (z : F), f (plus x y) z = plus (f x z) (f y z)) ∧
 (forall (x : E) (y z : F), f x (plus y z) = plus (f x y) (f x z)).
```

7.5.2 Norme subordonnée

La notion de norme subordonnée, dans le cas particulier des matrices, a déjà été abordée dans le chapitre 6 pour borner les erreurs d'arrondi des méthodes de Runge-Kutta. Il est en fait possible de généraliser ce concept à toute fonction d'un `NormedModule` dans un autre `NormedModule`.

Soit E, F : `NormedModule`, $f : E \to F$ et $\varphi : E \to$ `Prop`. Nous définissons la norme subordonnée de f sur φ, notée $|||f|||_\varphi$ et de type $(E \to F) \to \overline{\mathbb{R}}$, comme :

$$|||f|||_\varphi = \sup_{u \neq 0_E \wedge \varphi(u)} \frac{\|f(u)\|_F}{\|u\|_E}.$$

Dans le cas particulier où φ est exactement E, nous noterons simplement $|||f|||$.

Nous souhaitons définir cet objet comme une fonction totale. Or, il apparaît clairement qu'il faut traiter à part le cas où φ est le singleton $\{0_E\}$. En effet, dans ce cas particulier, l'ensemble $\{u \in E \mid u \neq 0_E \wedge \varphi(u)\}$ est vide et il faut choisir une valeur par défaut pour la norme d'opérateur : nous choisissons la valeur 0. Pour la disjonction de cas, il est nécessaire de décider si un sous-ensemble de E est égal à $\{0_E\}$, ce qui équivaut à décider si le sous-ensemble de $\mathbb{R}$ suivant est vide :

```
fun x:R ⇒ exists u:E, u <> zero ∧ phi u ∧ x = norm u.
```

Il est en fait toujours possible, dans la bibliothèque Coquelicot, de décider si un prédicat de type `R -> Prop` est vide ou non. Nous notons `Is_only_zero_set_dec_phi` le prédicat de décidabilité associé et formalisons la norme subordonnée de f sur φ :

```
Definition operator_norm_phi (f:E→ F) : Rbar :=
 match Is_only_zero_set_dec_phi E with
  | left _ ⇒ Lub_Rbar (fun x:R ⇒ exists u:E, u <> zero ∧
                        phi u ∧ x = norm (f u) / norm u)
  | right _ ⇒ 0
 end.
```

`Lub_Rbar P` est la construction de Coquelicot correspondant à la plus petite borne supérieure du sous-espace `P : Rbar -> Prop`.

7.5.3 Espace des fonctions linéaires continues

Dans le cas d'une fonction linéaire de $E \to F$, avec E et F munis de structures de `NormedModule`, nous prouvons l'équivalence entre huit définitions de la notion de continuité, comme proposé par Clément et Martin [50, théorème 145]. La définition de « référence » est la continuité au sens des filtres (voir section 7.1.2). Parmi les définitions équivalentes se trouvent des caractérisation *via* le caractère lipschitzien et *via* le caractère borné de la fonction. Une autre définition équivalente utilise la norme subordonnée. En effet, la proposition $\|\!|f|\!\| < +\infty$ caractérise également la continuité de f.

La preuve formelle se décompose en une série d'implications, certaines de ces implications ne nécessitant pas forcément l'hypothèse de linéarité. Selon l'usage, il peut être nécessaire d'utiliser l'une ou l'autre de ces définitions, *e.g.* lorsqu'il faut prouver la continuité d'une fonction, il est aisé de prouver qu'elle est bornée plutôt que de revenir à la définition utilisant les filtres.

Immler et Traut utilisent le caractère fini de la norme subordonnée comme caractérisation de la continuité des fonctions linéaires [117]. Nous optons pour le même choix et définissons l'ensemble des fonctions linéaires continues *via* le type dépendant `clm` :

```
Record clm := Clm {
  m:> E→F ;
  Lf: is_linear_mapping m;
  Cf: is_finite (operator_norm m) }.
```

Nous utiliserons la terminologie *composante fonctionnelle* pour désigner le champ `m` d'un élément de type `clm E F` (il s'agit de la fonction en elle-même, c'est pourquoi nous avons ajouté une coercion `m:> E->F`).

L'approche utilisant la norme subordonnée n'est pas choisie arbitrairement et permet de munir l'espace `clm E F` d'une structure d'`UniformSpace`, dont l'objet `ball` est induit à partir de la norme subordonnée. Nous montrons également qu'il peut être muni de la structure de `ModuleSpace`. Cela nécessite de prouver des égalités entre éléments de type `clm E F`, que l'axiome de `ProofIrrelevance` permet de réduire à l'égalité entre composantes fonctionnelles.

La norme subordonnée aux normes sur `E` et `F` n'est pas une norme pour l'ensemble des fonctions de `E -> F`. En effet, elle est à valeurs dans $\overline{\mathbb{R}}$ et non dans $\mathbb{R}$. Cependant, dans le cas particulier des fonctions linéaires continues, comme la norme subordonnée est finie, nous montrons que `clm E F` peut être muni de la structure de `NormedModule`.

Le cas particulier où `F` = $\mathbb{R}$ est très fréquemment utilisé dans le chapitre 8.

Définition 7.4 (Dual topologique).
Soit E : `NormedModule`. On appelle dual topologique de E l'ensemble E' = `clm E` $\mathbb{R}$.

7.5.4 Représentation des fonctions bilinéaires

Dans cette section, nous nous intéressons aux fonctions bilinéaires bornées, lesquelles interviennent dans l'énoncé du théorème de Lax–Milgram (théorème 8.4).

Définition 7.5 (Caractère borné d'une fonction bilinéaire).
Une fonction $f : E \to F \to \mathbb{R}$ *est bornée ssi*

$$\exists C \in \mathbb{R},\ \forall x \in E,\ \forall y \in F, \quad |f(x,y)| \leq C \left\|x\right\|_E \left\|y\right\|_F .$$

C *est appelée* constante de continuité *de* f.

Ensuite, nous démontrons un résultat-clef de représentation nécessaire à la preuve du théorème de Riesz-Fréchet (théorème 8.2) et assurant que toute forme bilinéaire bornée de E : `NormedModule` puisse être représentée par une fonction linéaire continue à valeurs dans le dual topologique de E :

Lemme 7.6. Représentation des formes bilinéaires bornées 
Soit $a : E \to E \to \mathbb{R}$. *Supposons que* a *est bilinéaire et bornée. Alors, il existe une unique fonction* $\mathcal{A}$ *de type* `clm` E E' *telle que*

$$\forall u, v \in E, \quad a(u,v) = (\mathcal{A}(u))(v).$$

Nous n'entrons pas dans les détails de la démonstration mais remarquons qu'elle présente quelques difficultés. En effet, après avoir exhibé la fonction $\mathcal{A}$, il faut d'une part prouver qu'elle est linéaire et continue, puis prouver que son application partielle à son premier argument est-elle même linéaire et continue (élément du dual topologique).

Le lemme 7.6 est la dernière brique d'une série de définitions et résultats génériques sur les espaces de Hilbert et les espaces de fonctions. Dans le chapitre suivant, nous présentons des résultats à la fois plus pointus et plus spécialisés, ayant pour objectif la formalisation du théorème de Lax–Milgram.

Chapitre 8

Théorèmes fondamentaux d'analyse fonctionnelle

La correction de la méthode des éléments finis repose sur le théorème de Lax–Milgram (théorème 8.4), un résultat fondamental d'analyse fonctionnelle [49, 64]. Il permet en effet d'établir l'existence et l'unicité de la solution du problème continu (*i.e.* l'équation aux dérivées partielles dont les solutions sont à valeurs dans un espace de Hilbert fonctionnel de dimension infinie) et de son approximation discrète sur un maillage de dimension finie. Cela ne suffit pas à montrer que la méthode des éléments finis fournit une bonne approximation de la solution exacte. Il faut pour cela utiliser un corollaire du théorème de Lax–Milgram appelé lemme de Céa (lemme 8.5). Le lemme de Céa fournit en effet une borne sur la différence entre la solution au problème continu et l'approximation de la solution sur le sous-espace de dimension finie. Le théorème de Lax–Milgram se voulant aussi générique que possible, il s'applique à tout sous-espace `ModuleSpace`-compatible et complet d'un espace de Hilbert E. Il faudra ensuite l'instancier à l'ensemble E tout entier (trivialement `ModuleSpace`-compatible et complet dans E) puis aux sous-espaces de dimension finie de E.

Nous aurions pu nous intéresser à d'autres résultats, jouant un rôle analogue au théorème de Lax–Milgram mais applicables dans un cadre plus général, comme le théorème de Banach–Nečas–Babuška [64, 42]. Notre choix s'est néanmoins porté sur le théorème de Lax–Milgram, qui permet d'étudier une classe suffisamment large d'équations différentielles et dont la preuve repose sur des fondements plus anciens et dont l'usage dans les démonstrations mathématiques est plus courant. S'ils ne sont pas nouveaux, ces fondements forment cependant un socle conséquent de concepts mathématiques, incluant les résultats formalisés dans le chapitre 7.

La démonstration du théorème de Lax–Milgram fait appel à d'autres résultats intermédiaires d'analyse fonctionnelle, à savoir un théorème de point fixe de Banach, présenté en section 8.1, ainsi qu'un résultat de représentation appelé théorème de Riesz–Fréchet, présenté en section 8.2. La formalisation de ces deux théorèmes en Coq est à la fois technique (définition de nouvelles structures de données, problèmes de décidabilité) et fastidieuse (raisonnements calculatoires). La démonstration du théorème de

Lax–Milgram est présentée en section 8.3.

L'application du théorème de Lax–Milgram aux sous-espaces de dimension finie est également difficile. En section 8.4, nous définissons tout d'abord la notion de sous-espace de dimension finie dans un espace de Hilbert quelconque, de dimension généralement infinie. Ensuite, nous démontrons la complétude des sous-espaces de dimension finie en section 8.5. La preuve de complétude fait intervenir de nombreux raisonnements topologiques, traités *via* des suites dans la majorité de la littérature mathématique [50]. Ainsi, pour pouvoir utiliser la bibliothèque Coquelicot, nous devons réinterpréter ces démonstrations du point de vue des filtres, ce qui pose de nombreux écueils techniques, comme la construction de transformations de filtres. La section 8.6 revient sur certains problèmes de décidabilité rencontrés dans le chapitre 7 et dans le présent chapitre.

8.1 Théorème de point fixe de Banach

En analyse fonctionnelle, et plus principalement dans les domaines liés à l'étude des équations différentielles, nombreux sont les résultats qui font appel à des théorèmes dits de « point fixe ». Leur point commun est d'établir l'existence (et parfois l'unicité) d'un point fixe pour une fonction donnée [90]. Les théorèmes de point fixe de Banach sont parmi les plus connus et sont utilisés dans les démonstrations de résultats fondamentaux, comme le théorème de Cauchy-Lipschitz [59] ou encore le théorème des fonctions implicites [194]. Le théorème de Lax–Milgram, démontré formellement en section 8.3, utilise une variante de théorème de point fixe de Banach.

Ce type de résultat étant très répandu, la formalisation de théorèmes de point fixe n'est pas nouvelle. Makarov et Spitters ont par exemple proposé une formalisation du théorème de Picard-Lindelöf en Coq, dont la preuve repose sur un théorème de point fixe de Banach [146]. Harrisson formalise les théorèmes de point fixe de Banach et de Brouwer en HOL [103]. Immler et Hölz ont également utilisé une variante du théorème de point fixe de Banach pour une formalisation d'équations différentielles en Isabelle/HOL [116]. La particularité du résultat auquel nous nous intéressons est le fait qu'il s'applique sur un sous-espace d'un espace complet. Pour des raisons de lisibilité, nous commençons par l'énoncer en langage mathématique, sans considérer la notion de filtre (pour rappel, voir la définition des boules généralisées en section 7.1.2) :

Théorème 8.1. Point fixe de Banach 

Soit E : `CompleteSpace` *et* $\varphi : E \to$ `Prop` *un sous-espace non vide et complet de* E. *Soit* $f : E \to E$. *Soit* $(x_n)_{n\in\mathbb{N}}$ *définie par* $x_0 \in \varphi$ *et* $\forall n, x_{n+1} = f(x_n)$. *Supposons que :*

- $\forall x, y, \varphi(x) \Rightarrow \varphi(y) \Rightarrow \exists M, 0 \leqslant M \wedge y \in \mathcal{B}(x, M)$;
- f *est contractante ;*
- $\forall x, \varphi(x) \Rightarrow \varphi(f(x))$.

Alors :

- $\exists! a \in E, \varphi(a) \wedge a = f(a)$;
- $(x_n)_{n\in\mathbb{N}}$ *est convergente et* $\lim_{x\to\infty} f(x) = a$.

En d'autres termes, toute fonction f contractante sur un sous-espace φ non vide et complet d'un espace complet E est telle que ses itérées successives à partir d'un point $x_0 \in \varphi$ convergent vers un unique point fixe $a \in \varphi$.

Une première étape consiste à définir la notion de fonction contractante (et donc de fonction lipschitzienne) sur une structure algébrique qui n'est équipée ni de norme, ni même de métrique. Dans une structure d'`UniformSpace`, seules les boules généralisées (pseudométriques) sont disponibles. Considérons une fonction `f : X -> Y` avec `X, Y : UniformSpace`, nous notons `ball_x` et `ball_y` les boules généralisées associées respectivement à `X` et `Y`. Nous définissons le caractère k-lipschitzien (puis contractant lorsque $k < 1$) de `f` de la manière suivante :

```
Definition is_Lipschitz (f : X → Y) (k : R) :=
  0 <= k ∧ forall x1 x2 r, 0 < r → ball_x x1 r x2
                                → ball_y (f x1) (k*r) (f x2).

Definition is_contraction (f : X → Y) :=
  exists k, k < 1 ∧ is_Lipschitz f k.
```

Les notions topologiques définies dans Coquelicot utilisent les filtres. Il est donc nécessaire d'exhiber un filtre jouant un rôle analogue à la suite $(x_n)_{n\in\mathbb{N}}$ de l'énoncé du théorème 8.1. À cet effet, nous définissons le filtre suivant, qui est propre et de Cauchy :

```
F := (fun P ⇒ eventually (fun n ⇒ P (iter f n x0)))
```

avec `x0` un élément du sous-ensemble `phi`.

L'énoncé Coq correspondant au théorème 8.1 est le suivant (voir la définition de `close` en section 7.1.2 et la définition de `my_complete` en section 7.2.2) :

```
Context {E : CompleteSpace}.

Hypothesis phi_f : forall x : E, phi x → phi (f x)
Hypothesis phi_distanceable: forall (x y:E),
     phi x → phi y → exists M, 0 <= M ∧ ball x M y.
Hypothesis phi_complete : my_complete phi.
Hypothesis phi_not_empty : exists a : E, phi a

Theorem FixedPoint_C_phi : is_contraction_phi f phi →
  exists a:E, phi a ∧ close (f a) a
         ∧ (forall b, phi b → close (f b) b → close b a)
         ∧  forall x, phi x → close (lim F) x.
```

Outre le fait que le filtre `F` soit utilisé en lieu et place de la suite $(x_n)_{n\in\mathbb{N}}$, nous remarquons de légères différences avec l'énoncé mathématique « standard ». En effet, le résultat démontré relâche la notion d'égalité, utilisant plutôt la propriété `close` des `UniformSpace`. La conclusion du théorème assure l'existence d'un élément `a` vérifiant une conjonction de 4 propositions : la première exprime l'appartenance de `a` à `phi` ; la

seconde fait de cet élément un point fixe pour la fonction `f` ; la troisième proposition assure l'unicité du point fixe ; et la quatrième lie le point fixe à la limite du filtre `F`.

Nous n'utiliserons en fait ce résultat que pour `E : Hilbert` et *a fortiori* pour `E` muni d'une structure de `CompleteNormedModule`, pour laquelle il est possible de prouver que `forall x y, close x y <-> x = y`. Par ailleurs, l'hypothèse `phi_distanceable` deviendra inutile car la propriété suivante est toujours vérifiée :

$$\forall x, y \in E, y \in \mathcal{B}(x, 2 \times \|x - y\|)$$

8.2 Théorème de Riesz–Fréchet

Un résultat intermédiaire permettant de démontrer le théorème de Lax–Milgram est le théorème de Riesz–Fréchet. Il s'agit d'un résultat de représentation au sens où il permet de représenter les éléments du dual topologique (voir définition 7.4) d'un espace de Hilbert comme produit scalaire par un vecteur bien choisi de l'espace.

8.2.1 Noyau d'une fonction

Afin de pouvoir énoncer le théorème de Riesz–Fréchet, il faut commencer par formaliser la notion de noyau de fonction. Considérons $f : E \to F$ avec E, F : `ModuleSpace`. Le noyau de f, noté $\ker(f)$, est le sous-espace de E sur lequel f s'annule, *i.e.* :

```
Definition ker (f : E →F) := fun x:E ⇒f x = zero.
```

Si $f : E'$ (élément du dual topologique de E, voir définition 7.4), nous montrons que $\ker(f)$ est un sous-ensemble `ModuleSpace`-compatible et fermé (voir définition de `closed` en section 7.2.2) de E.

La `ModuleSpace`-compatibilité se démontre facilement. En revanche, la preuve de fermeture de $\ker(f)$ est plus intéressante. L'argument sur lequel repose la démonstration est la continuité de f. En effet, $\ker(f) = f^{-1}(\{0_F\})$; or tout singleton est fermé et *a fortiori* $\{0_F\}$ est fermé. Par continuité de f, l'ensemble $f^{-1}(\{0_F\})$ est également fermé.

8.2.2 Énoncé du théorème

Théorème 8.2. Riesz–Fréchet  $_{P \vee \neg P}$
Soit E : `Hilbert`, $f \in E'$, $\varphi : E \to$ `Prop`, φ `ModuleSpace`-compatible et complet. Alors, il existe un unique élément $u \in E$ tel que :

$$\varphi(u) \quad \wedge \quad \forall v \in E,\ \varphi(v) \implies f(v) = \langle u, v \rangle_E .$$

La preuve d'unicité est relativement naïve et est traitée indépendamment de la preuve d'existence. La preuve d'existence nécessite de disjoindre le cas où la forme linéaire f est identiquement nulle sur le sous-ensemble φ et le cas où elle ne l'est pas[1].

1. En logique intuitionniste, cela pose des problèmes de décidabilité, voir section 8.6.

Si f est identiquement nulle sur φ, il est évident que 0 est solution du problème puisque pour tout v, $f(0) = \langle 0, v\rangle_E = 0$.

Dans le second cas, nous exhibons un élément $u_0 \in E$ tel que $f(u_0) \neq 0$, *i.e.* $u_0 \in \varphi \cap (\ker(f))^\perp$. Comme $\ker(f)$ est un sous-ensemble fermé et donc complet de E, d'après le théorème 7.3, il existe un unique projeté orthogonal $P_{\ker(f)}(u_0)$ de u_0 sur $\ker(f)$. La suite de la démonstration consiste à construire une solution pour satisfaire le problème posé par le théorème de Riesz–Fréchet. Cette solution est construite pas à pas à partir du vecteur $u_0 - P_{\ker(f)}(u_0)$, qui est la projection orthogonale de u_0 sur $(\ker(f))^\perp$. Le vecteur $u_0 - P_{\ker(f)}(u_0)$ est régulièrement utilisé comme dénominateur de fractions dans la construction de la solution, ce qui implique de démontrer que $u_0 - P_{\ker(f)}(u_0) \neq 0$. Pour cela, nous montrons $P_{(\ker(f))^\perp}(u_0) = u_0 - P_{\ker(f)}(u_0)$, *i.e.* $u_0 - P_{\ker(f)}(u_0) \in (\ker(f))^\perp$, *i.e.* $u_0 - P_{\ker(f)}(u_0) \neq 0$. L'ensemble de la construction peut être consulté dans [50, théorème 199] ou dans le fichier Coq `lax_milgram.v`.

8.2.3 Fonction de représentation de Riesz

Pour chaque forme linéaire continue $f \in E'$, le théorème de Riesz–Fréchet assure l'existence d'une unique solution u au problème $\forall v \in E,\ \varphi(v) \implies f(v) = \langle u, v\rangle_E$. Cette preuve d'existence ne fournit pas une fonction calculatoire qui, sur la donnée de f, renvoie l'unique solution associée. La démonstration de Clément et Martin utilise une fonction $\tau : E' \to E$ à cet effet [50] :

Définition 8.1 (**Fonction de représentation de Riesz**).
Soit E : `Hilbert`. D'après le théorème 8.2, pour tout $f \in E'$, il existe un unique $u_f \in E$ tel que $\forall v \in E,\ f(v) = \langle u_f, v\rangle_E$. La fonction de représentation de Riesz, notée τ, est la fonction définie par $\forall f \in E', \tau(f) = u_f$.

La fonction τ ne peut pas être définie de manière purement constructive. Nous pouvons néanmoins utiliser l'opérateur `iota` de Coquelicot, déjà introduit et utilisé en section 7.4 pour construire le projeté orthogonal, qui peut être défini à partir de la seule axiomatisation des réels de la bibliothèque standard. Nous formalisons τ comme une fonction totale sur le dual topologique d'un espace de Hilbert E :

```
Definition tau := fun (f : topo_dual E) ⇒
  (iota (fun u : E ⇒ forall v : E, f v = inner u v)).
```

La preuve du théorème de Lax–Milgram utilise deux propriétés de τ :

Lemme 8.3. Propriétés de la fonction de représentation de Riesz  $_{P \vee \neg P}$

- τ *est une fonction linéaire ;*
- $\forall f \in E',\ |||f||| = \|\tau(f)\|_E$.

Si la preuve de linéarité est relativement simple, la preuve d'isométrie (*i.e.* $\forall f \in E',\ |||f||| = \|\tau(f)\|_E$) est assez fastidieuse. En effet, nous utilisons l'opérateur `glb_Rbar` (plus grande borne inférieure) de Coquelicot, qui est à valeurs dans $\overline{\mathbb{R}}$ dans le cas général. Or, en Coq, l'utilisation de $\overline{\mathbb{R}}$ rend généralement les preuves fastidieuses puisqu'il est nécessaire de traiter à part les cas $+\infty$ et $-\infty$.

8.3 Théorème de Lax–Milgram

Le théorème de Lax–Milgram repose sur deux propriétés des fonctions à deux arguments : le caractère borné (définition 7.5) et la coercivité (définie ci-dessous).

Définition 8.2 (Coercivité).
Soit E : `Hilbert`, $\alpha \in \mathbb{R}_+^*$. *Une fonction* $f : E \to E \to \mathbb{R}$ *est* α*-coercive ssi* $\forall x \in E, \alpha \|x\|_E^2 \leq f(x,x)$.

S'en suit l'énoncé du théorème de Lax–Milgram :

Théorème 8.4. Lax–Milgram $_{P\vee\neg P}$
Soit E : `Hilbert`, $f \in E'$, $C, \alpha \in \mathbb{R}_+^*$. *Soit* $\varphi : E \to$ `Prop`, φ `ModuleSpace`*-compatible et complet. Soit* a *une forme bilinéaire de* E *bornée par* C *et* α*-coercive. Alors :*

$$\exists! u \in E, \varphi(u) \wedge \forall v \in E, \varphi(v) \Longrightarrow f(v) = a(u,v) \wedge \|u\|_E \leq \frac{1}{\alpha} |||f|||_\varphi.$$

Démonstration. La démonstration du théorème de Lax–Milgram est en partie calculatoire. Il faut en effet prouver de nombreuses inégalités ou égalités entre expressions mathématiques. Cela rend la formalisation des résultats très fastidieuse. Afin d'obtenir une preuve formelle relativement courte et compréhensible, nous avons choisi de traiter à part ces raisonnements calculatoires, *via* un ensemble de lemmes auxiliaires.

Le coeur technique de la preuve s'articule en trois étapes. La première étape consiste à montrer que le problème posé par le théorème de Lax–Milgram, *i.e.* $\mathcal{P}_1 \equiv \exists! u \in E, \varphi(u) \wedge \forall v \in E, \varphi(v) \Longrightarrow f(v) = a(u,v)$, se réduit à la résolution d'un problème $\mathcal{P}_2$ plus simple. La seconde étape consiste à montrer que la résolution de ce problème $\mathcal{P}_2$ peut se réduire à l'exhibition de l'unique point fixe d'une fonction bien choisie (problème $\mathcal{P}_3$). Enfin, la troisième étape de la démonstration consiste à résoudre le problème de point fixe $\mathcal{P}_3$.

Étape 1 : Réduction à un problème plus simple. D'après le lemme 7.6, il existe une unique forme linéaire continue $\mathcal{A}$ de $E \to E'$ telle que :

$$\forall u, v \in E,\ (\mathcal{A}(u))(v) = a(u,v).$$

Ensuite, on utilise le théorème de Riesz–Fréchet (théorème 8.2) à deux reprises (sur $\mathcal{A}(u)$ et sur f) afin d'exhiber les deux résultats suivants :

$$\exists! w_{a_u} \in E, \forall v \in E,\ a(u,v) = (\mathcal{A}(u))(v) = \langle w_{a_u}, v \rangle_E,$$
$$\exists! w_f \in E, \forall v \in E,\ f(v) = \langle w_f, v \rangle_E.$$

Ainsi, en posant $\tau(\mathcal{A}(u)) = w_{a_u}$ et $\tau(f) = w_f$, nous obtenons successivement :

$$\forall v \in E, \langle w_{a_u}, v \rangle_E = \langle w_f, v \rangle_E \Longrightarrow \forall v \in E, a(u,v) = f(v),$$

et

$$w_{a_u} = w_f \Rightarrow \forall v \in E,\ a(u,v) = f(v),$$
$$\tau(\mathcal{A}(u)) = \tau(f) \Rightarrow \forall v \in E,\ a(u,v) = f(v).$$

Il est donc possible de réduire le problème $\mathcal{P}_1$ à un problème $\mathcal{P}_2$ plus simple à travers l'implication suivante :

$$\underbrace{\exists! u \in E, \tau(\mathcal{A}(u)) = \tau(f)}_{\mathcal{P}_2} \Longrightarrow \underbrace{\exists! u \in E,\ \forall v \in E, a(u,v) = f(v)}_{\mathcal{P}_1}. \tag{8.1}$$

Étape 2 : Réduction à un problème de point fixe. Soit τ la fonction de représentation de Riesz (voir définition 8.1). Soit $\rho \in \mathbb{R}$ tel que $0 < \rho < \frac{2\alpha}{C^2}$. Nous définissons $g : E \to E$ par :

$$\forall x \in E,\ g(x) = x - \rho \cdot \tau(A(x)) + \rho \cdot \tau(f).$$

Nous montrons ensuite, de façon assez naïve et en dépliant la définition de g, l'implication qui suit :

$$g(u) = u \Longrightarrow \tau(\mathcal{A}(u)) = \tau(f).$$

Nous en déduisons de façon immédiate :

$$\underbrace{\exists! u \in E, g(u) = u}_{\mathcal{P}_3} \Longrightarrow \underbrace{\exists! u \in E, \tau(\mathcal{A}(u)) = \tau(f)}_{\mathcal{P}_2}. \tag{8.2}$$

Étape 3 : Résolution du problème de point fixe. Par un raisonnement calculatoire, nous prouvons que g est k-Lipschitzienne avec $k = \sqrt{1 - 2\rho\alpha + \rho^2 C^2}$ et prouvons que pour tout ρ tel que $0 < \rho < \frac{2\alpha}{C^2}$, nous avons $0 < k < 1$, faisant ainsi de g une fonction contractante. Enfin, par application du théorème de point fixe de Banach (théorème 8.1), il existe un unique point fixe pour g.

Conclusion. Donc, *via* (8.2) et (8.1), nous démontrons le théorème de Lax–Milgram. □

Le théorème de Lax–Milgram peut être appliqué à la fois sur un espace de Hilbert E et sur un sous-ensemble `ModuleSpace`-compatible et complet de E. Nous obtenons donc deux uniques solutions dont il est possible de borner la différence :

Lemme 8.5. Céa 

Soit E : `Hilbert`, $f \in E'$, $0 < \alpha$. Soit $\varphi : E \to$ `Prop`, φ `ModuleSpace`-compatible et complet. Soit a une forme bilinéaire de E, bornée par $C > 0$ et α-coercive. Soit u et u_φ les solutions données par application du théorème de Lax–Milgram respectivement sur E et sur le sous-ensemble φ. Alors :

$$\forall v_\varphi \in E, \varphi(v_\varphi) \Longrightarrow \|u - u_\varphi\|_E \leq \frac{C}{\alpha} \|u - v_\varphi\|_E.$$

La démonstration du lemme de Céa est presque immédiate, la difficulté principale résidant dans la preuve du théorème de Lax–Milgram sans lequel il ne peut être énoncé. Ce lemme est néanmoins indispensable pour établir la convergence de la méthode des éléments finis. Pour ce faire, il faut l'appliquer à un sous-espace φ de dimension finie. Il faut donc formaliser la notion de sous-espace de dimension finie d'un espace de Hilbert et démontrer qu'un tel sous-espace est `ModuleSpace`-compatible et complet.

8.4 Sous-espaces de dimension finie

Dans cette section, nous présentons une formalisation des sous-espaces de dimension finie des espaces de Hilbert. Contrairement à la majorité des travaux présentés en section 3.2.3, nous définissons des sous-espaces de dimension finie d'espaces de dimension potentiellement infinie (espaces de Hilbert), ce qui rend leur formalisation plus difficile.

Un sous-espace φ de E : `Hilbert` est de dimension finie n et famille génératrice B s'il vérifie la propriété `FDIM` définie ci-dessous :

```
Definition FDIM (phi : E → Prop) (n : nat) (B : nat → E) :=
    match (eq_nat_dec n 0) with  (* teste si la dimension est nulle *)
     | left _ ⇒ forall u, phi u ↔ u = zero              (* n = 0 *)
     | right _ ⇒ forall u, phi u ↔ exists L : nat → R, (* n > 0 *)
           u = sum_n (fun k ⇒ scal (L k) (B k)) (n-1)
end.
```

En d'autres termes, seul l'ensemble réduit au singleton $\{0\}$ est de dimension 0 et pour $n > 0$, les vecteurs u d'un espace de dimension n sont combinaisons linéaires des n premiers vecteurs de la famille génératrice B, *i.e.* $u = \sum_{k=0}^{n-1} L_k B_k$ avec $L : \mathbb{N} \to \mathbb{R}$.

Faisons quelques remarques sur ce choix de formalisation :

- Tout d'abord, B est une famille génératrice du sous-espace de dimension finie, et non une base. En effet, B n'est pas forcément une famille libre, *i.e.* telle que $\forall L \in \mathbb{R}^n, \sum_{k=0}^{n-1} L_k B_k = 0 \Rightarrow (\forall i, 0 \leqslant i < n \Rightarrow L_i = 0)$.
- De plus, nous ne construisons pas B comme une liste de taille n mais comme une suite de vecteurs. Cela simplifie l'initialisation de B car pour $p \geqslant n$, nous n'avons pas à nous soucier de la valeur de B_p, qui n'est pas nécessairement nulle.
- En outre, n est une surestimation de la dimension : nous pourrions imaginer qu'il existe des vecteurs B_j de la famille génératrice tels que $j < n$ et $B_j = 0$.

Une famille génératrice B est dite orthonormée si elle vérifie la propriété suivante :

```
Definition B_ortho (B : nat → E) := forall i, inner (B i) (B i) = 1
                        ∧ forall i j, i <> j → inner (B i) (B j) = 0.
```

Supposer que la famille génératrice d'un sous-espace de dimension finie est orthonormée simplifie considérablement les démonstrations. En fait, il existe un moyen canonique et constructif de transformer n'importe quelle famille génératrice en une famille génératrice orthonormée. Cette construction est connue sous le nom de procédé d'orthonormalisation de Gram-Schmidt [123]. Nous n'avons pas formalisé le procédé d'or-

thonormalisation de Gram-Schmidt en Coq mais supposons que la famille génératrice est orthonormée quand cela s'avère nécessaire ou que cela simplifie les démonstrations.

Nous montrons tout d'abord la `ModuleSpace`-compatibilité des sous-espaces de dimension finie, première propriété nécessaire à l'application du théorème de Lax–Milgram :

Lemme 8.6. `ModuleSpace`-compatibilité, sous-espaces de dimension finie 
Soit E : `Hilbert`, $\varphi : E \to Prop$. Supposons que φ est de dimension finie (la famille génératrice n'est pas nécessairement orthonormée). Alors φ est `ModuleSpace`-compatible.

La démonstration du lemme 8.6 est relativement naïve. En revanche, la propriété de complétude est plus difficile à démontrer.

8.5 Propriétés de complétude

Cette section est vouée à la preuve de complétude des sous-espaces de dimension finie des espaces de Hilbert. Dans la section 8.5.1, nous introduisons le concept de sous-module engendré par un vecteur et montrons que ces sous-espaces sont complets. Puis, en section 8.5.2, nous utilisons le fait qu'un sous-espace de dimension finie soit somme directe des sous-modules engendrés par les vecteurs de sa famille génératrice pour démontrer sa complétude.

8.5.1 Complétude des sous-modules engendrés

Le sous-module engendré par un vecteur $u \in E$ (E : `Hilbert`) est l'ensemble des vecteurs de E qui sont colinéaires à u. Le terme Coq associé est noté `span u` :

```
Definition span (u : E) := fun (x : E) ⇒ exists (l : R), x = scal l u.
```

Avant de pouvoir démontrer la complétude des sous-espaces de dimension finie, il est nécessaire de prouver la complétude des sous-modules engendrés.

Lemme 8.7. Complétude des sous-modules engendrés  $_{P \vee \neg P}$
Soit E : `Hilbert`, $u \in E$. Alors span(u) *est complet.*

Idées de la preuve papier : La preuve papier du lemme 8.7 proposée par Clément et Martin montre la propriété de complétude en faisant converger des suites de Cauchy [50, lemme 115]. Pour être plus précis, Clément et Martin montrent que toute suite de Cauchy de la forme $(\lambda_n u)_{n\in\mathbb{N}}$ (*i.e.* une suite à valeurs dans span(u)) converge dans span(u). Pour cela, ils commencent par extraire la suite $(\lambda_n)_{n\in\mathbb{N}}$, montrent que c'est également une suite de Cauchy et que, par complétude de $\mathbb{R}$, elle converge vers $\ell \in \mathbb{R}$. Enfin, ils en déduisent que $(\lambda_n u)_{n\in\mathbb{N}}$ converge vers ℓu qui est bien à valeurs dans span(u). La preuve *via* les suites est donc relativement naturelle.

Formalisation de la preuve et passage aux filtres : L'utilisation des filtres, conjuguée à la manière dont sont traités les sous-espaces, rend particulièrement ardue la formalisation de cette démonstration.

Une première difficulté est de dire d'un filtre qu'il est « un filtre sur $\mathrm{span}(u)$ » (par analogie avec la suite $(\lambda_n u)_{n\in\mathbb{N}}$ dans la preuve de Clément et Martin). Comme évoqué dans la section 7.2.2 , nous ne pouvons pas construire explicitement un filtre sur le sous-espace $\mathrm{span}(u)$. Nous utilisons donc le concept de filtre induit (définition 7.1) et sommes ramenés à prouver que tout filtre propre et de Cauchy qui est induit sur $\mathrm{span}(u)$ converge dans $\mathrm{span}(u)$.

Une seconde difficulté réside dans l'extraction du filtre des coefficients multiplicatifs, c'est-à-dire le filtre analogue à la suite $(\lambda_n)_{n\in\mathbb{N}}$ dans la preuve de Clément et Martin. En effet, il n'est pas évident d'extraire un tel filtre à partir d'un filtre induit sur $\mathrm{span}(u)$. À cet effet, nous définissons une fonction totale sur les filtres de E dont l'image est un filtre sur $\mathbb{R}$ et que nous appelons *opérateur de nettoyage scalaire* :

```
Definition cleansc (u : E) (F : (E → Prop) → Prop) : (R → Prop) → Prop
           := fun A ⇒ exists V, F V ∧ (forall k, V (scal k u) → A k).
```

Considérons le cas particulier où le filtre $\mathscr{F}$ est induit sur $\mathrm{span}(u)$. Par définition des filtres induits, nous savons que $\forall\varphi, \mathscr{F}(\varphi) \Rightarrow \exists x, x \in \varphi \cap \mathrm{span}(u)$ (*i.e.* $x = \lambda u$ pour un certain coefficient multiplicatif λ). Pour chaque sous-ensemble contenu dans $\mathscr{F}$, il existe un tel coefficient multiplicatif. Le filtre `cleansc u` $\mathscr{F}$ est le filtre contenant l'ensemble de ces coefficients multiplicatifs.

En revanche, dans le cas où le filtre en entrée n'est pas induit sur $\mathrm{span}(u)$, `cleansc u` renvoie une valeur arbitraire et qui n'a aucune importance (car inutilisée).

Si $\mathscr{F}$ est induit sur $\mathrm{span}(u)$, propre et de Cauchy, nous prouvons que `cleansc u` $\mathscr{F}$ est également propre et de Cauchy. Cela permet d'utiliser la complétude de $\mathbb{R}$ et d'extraire une limite ℓ pour `cleansc u` $\mathscr{F}$. Nous montrons ensuite que la limite de $\mathscr{F}$ ne peut qu'être égale à ℓu en utilisant un argument d'unicité sur les limites. Cette limite est trivialement un élément de $\mathrm{span}(u)$, ce qui achève la démonstration du lemme 8.7.

8.5.2 Complétude des sous-espaces de dimension finie

Nous démontrons dans un premier temps que la somme directe d'un sous-ensemble complet ψ et du sous-module engendré par un vecteur de $\psi^\perp$ est complet.

Lemme 8.8. Complétude, somme directe et sous-module engendré  $_{P\vee\neg P}$
Soit E : Hilbert, $\varphi, \psi : E \to Prop$, $u \in E$. Supposons que ψ est complet, que $\varphi = \psi \oplus \mathrm{span}(u)$ et que $u \in \psi^\perp$. Alors φ est complet.

Une démonstration à base de suites est proposée par Clément et Martin [50, lemme 196]. Cette démonstration étant longue, nous ne donnons que quelques intuitions puis mettons en exergue les difficultés liées au passage vers une preuve à base de filtres.

Idées de la preuve papier : Soit $(w_n)_{n\in\mathbb{N}}$ une suite de Cauchy à valeurs dans φ. *Via* l'argument de somme directe, nous pouvons décomposer $(w_n)_{n\in\mathbb{N}}$ sur φ et span(u), *i.e.* $(w_n)_{n\in\mathbb{N}} = (v_n + \lambda_n u)_{n\in\mathbb{N}}$ avec $(v_n)_{n\in\mathbb{N}}$ une suite à valeurs dans ψ et $(\lambda_n u)_{n\in\mathbb{N}}$ à valeurs dans span(u). Il est en fait possible, en utilisant le lemme 7.5, la linéarité du projeté orthogonal et l'appartenance de u à $\psi^\perp$, de montrer que $v_n = P_\psi(w_n)$ puis que $P_{\psi^\perp}(w_n) = \lambda_n u$. Nous prouvons ensuite la continuité de P_ψ et $P_{\psi^\perp}$, qui permet de démontrer que $(w_n)_{n\in\mathbb{N}}$ converge dans φ.

Formalisation de la preuve et passage aux filtres : De même que pour la démonstration du lemme 8.7, la difficulté du passage aux filtres réside principalement dans l'extraction de filtres analogues aux suites $(v_n)_{n\in\mathbb{N}}$ et $(\lambda_n u)_{n\in\mathbb{N}}$ à partir d'un filtre induit sur φ (analogue à la suite $(w_n)_{n\in\mathbb{N}}$). Néanmoins, grâce à la caractérisation utilisant les projetés orthogonaux, nous définissons deux projecteurs permettant d'extraire ces filtres en utilisant l'opérateur `filterlim` de Coquelicot (voir section 7.1.1) :

```
Definition proj_filter_ortho (psi : E → Prop) (F : (E → Prop) → Prop)
                          := filtermap (proj psi) F.

Definition proj_filter_ortho_compl (psi : E → Prop)
      (F : (E → Prop) → Prop) := filtermap (fun x ⇒ x - proj psi x) F.
```

La complétude de ψ conditionne l'existence et l'unicité du projeté orthogonal et est donc intrinsèquement nécessaire à la définition des opérateurs `proj_filter_ortho` et `proj_filter_ortho_compl`.

Quand $\mathscr{F}$ est propre, de Cauchy et induit sur φ, nous prouvons que les filtres `proj_filter_ortho` $\mathscr{F}$ et `proj_filter_ortho_compl` $\mathscr{F}$ sont propres et de Cauchy. De plus, nous montrons que `proj_filter_ortho` $\mathscr{F}$ et `proj_filter_ortho_compl` $\mathscr{F}$ sont respectivement induits sur ψ et span(u).

Supposons que $\mathscr{F}$ converge vers une limite ℓ. Alors, par continuité de P_ψ et $P_{\psi^\perp}$, les limites respectives de `proj_filter_ortho` $\mathscr{F}$ et `proj_filter_ortho_compl` $\mathscr{F}$ sont $P_\psi(\ell)$ et $P_{\psi^\perp}(\ell) = \ell - P_\psi(\ell)$.

Comme span(u) est complet (lemme 8.7) et que `proj_filter_ortho_compl` $\mathscr{F}$ est propre, de Cauchy et induit sur span(u), il converge dans span(u) et donc il existe un élément $\lambda \in \mathbb{R}$ tel que $P_{\psi^\perp}(\ell) = \lambda u$. Donc, $\ell = P_\psi(\ell) + P_{\psi^\perp}(\ell) = P_\psi(\ell) + \lambda u$, *i.e.* $\ell \in \psi \oplus \text{span}(u)$, *i.e.* $\ell \in \varphi$, ce qui conclut la démonstration.

Nous pouvons maintenant énoncer et démontrer le résultat final de ce chapitre.

Théorème 8.9. Complétude des sous-espaces de dimension finie  $_{P\vee\neg P}$
Soit E : `Hilbert` et $\varphi : E \to$ Prop un sous-ensemble de dimension finie de E dont la famille génératrice est orthonormée. Alors φ est complet.

Démonstration. Soit n la dimension de φ et B la famille génératrice orthonormée de φ. Nous raisonnons par induction sur n :

- Si $n = 0$, $\varphi = \{0\}$. Comme tout singleton est fermé, et que tout sous-espace fermé d'un espace complet est complet, φ est complet.
- Si $n = m + 1$, comme φ est de dimension finie :

$$\forall u \in \varphi, \exists L : \mathbb{N} \to \mathbb{R}, u = \sum_{i=0}^{m} L_i B_i = \sum_{i=0}^{m-1} L_i B_i + L_m B_m.$$

Par ailleurs, comme B est orthonormée, cette décomposition est unique, et donc :

$$\exists \psi : E \to Prop, \varphi = \psi \oplus \text{span}(B_m).$$

De plus, ψ est de dimension finie $m = n - 1$ et a pour famille génératrice $B^\psi = B$. Comme B^ψ est également orthonormée, par hypothèse d'induction, ψ est complet. De plus, comme B est orthonormée, $B_m \in \psi^\perp$. Donc, par application du théorème 8.8, $\varphi = \psi \oplus \text{span}(B_m)$ est complet.

□

8.6 Questions liées à la décidabilité

Certains théorèmes présentés dans ce chapitre (ainsi que dans le chapitre 7) sont dits classiques car leur démonstration formelle fait intervenir l'axiome du tiers-exclu, *i.e.* pour toute proposition P, $P \vee \neg P$. Il n'y a en fait nul besoin de supposer $P \vee \neg P$ vraie pour tout P. Il suffit de supposer vraies quelques hypothèses de décidabilité pour des propositions bien choisies dès que cela s'avère nécessaire. Avec l'appui de Coq, nous avons pu identifier et isoler l'ensemble des hypothèses de décidabilité nécessaires aux démonstrations et qui ne peuvent être obtenues en logique intuitionniste.

Plusieurs raisons ont motivé notre choix de ne pas présenter ces hypothèses au fil des preuves. D'une part, comme nous utilisons une axiomatisation des nombres réels, nous ne pouvons pas extraire de contenu calculatoire à partir de nos démonstrations. Il serait d'ailleurs vain de croire que nos démonstrations s'inscrivent dans le sillon de l'analyse fonctionnelle constructive, qui est par nature liée aux nombres réels constructifs [193]. Par ailleurs, certaines hypothèses de décidabilité sont difficiles à énoncer et leur justification présente peu d'intérêt.

Dans cette section, nous avons sélectionné quelques hypothèses de décidabilité que nous avons jugé utiles de présenter [2].

8.6.1 Double négation en logique intuitionniste

L'introduction et l'élimination de la double négation sont des règles de logique propositionnelle permettant de lier une proposition P à la négation de sa négation $\neg\neg P$.

En logique classique, il est possible de démontrer l'équivalence $P \Leftrightarrow \neg\neg P$. Les règles d'introduction ($P \Rightarrow \neg\neg P$) et d'élimination ($\neg\neg P \Rightarrow P$) de la négation sont donc toutes deux valides. En revanche, en logique intuitionniste, seule la règle d'introduction est

2. Néanmoins, l'intégralité des hypothèses ont été identifiées dans la formalisation Coq des résultats.

valide. Autrement dit, la proposition $\forall P, \neg\neg P \Rightarrow P$ n'est pas prouvable en logique intuitionniste, d'où la présence de doubles négations dans certains énoncés Coq.

8.6.2 Sous-espaces et plus grande borne inférieure

Le premier résultat du manuscrit identifié comme classique est le théorème 7.3. Ce résultat d'existence du projeté orthogonal sur un sous-espace `ModuleSpace`-compatible et complet $\varphi : E \to Prop$ utilise en effet l'hypothèse suivante :

```
Hypothesis H1 : forall u : E, forall eps : posreal,
          decidable (exists w : E, ¬¬phi w ∧ norm (minus u w) < eps).
```

Cette hypothèse permet de décider s'il existe ou non un vecteur de $\neg\neg\varphi$ qui soit suffisamment proche d'un autre point donné de E. Dans la démonstration du théorème 7.3, nous devons exhiber, pour $u \in E$ donné et pour un $\varepsilon \in \mathbb{R}_+^*$ arbitrairement petit, un élément x vérifiant $\neg\neg\varphi(x) \quad \wedge \quad \|u - x\|_E < \inf_{w \in E \wedge \varphi(w)} \|u - w\|_E + \varepsilon$.

L'existence d'une suite convergente dans $\mathbb{R}$ dont la limite est la plus grande borne inférieure de l'ensemble $\{\|u - w\|_E \mid w \in E \wedge \varphi(w)\}$ semble naturelle. Néanmoins, la définition de plus grande borne inférieure dans la bibliothèque Coquelicot rend possible la démonstration du fait « qu'il est faux d'attester qu'une telle suite n'existe pas » mais ne permet pas de prouver l'existence d'une telle suite. C'est pourquoi il est nécessaire de supposer vraie l'hypothèse `H1`.

En logique intuitionniste, le terme `¬ ¬ phi x` de l'hypothèse `H1` ne peut pas être remplacé par `phi x` en toute généralité (voir section 8.6.1). En revanche, cette simplification est possible lorsque `phi` est décidable[3].

L'hypothèse de décidabilité `H1` doit par ailleurs être propagée à l'ensemble des résultats qui dépendent du théorème 7.3, faisant ainsi d'eux des théorèmes dits classiques.

8.6.3 Décidabilité de la non-nullité d'une fonction

La démonstration du théorème 8.2 (Riesz–Fréchet) repose sur deux hypothèses de décidabilité. La première dérive directement de l'hypothèse `H1` définie ci-dessus.

Par ailleurs, la démonstration repose sur une disjonction de cas selon que la fonction f du dual topologique soit identiquement nulle ou non sur un sous-espace φ. Lorsque la fonction n'est pas identiquement nulle, il est nécessaire d'extraire u_0 tel que $f(u_0) \neq 0$.

Or, il n'est pas possible de décider constructivement si une fonction est identiquement nulle ou non car cela nécessiterait de tester l'égalité à 0 pour tous les points du sous-espace $\neg\neg\varphi$, *i.e.* pour un ensemble potentiellement infini de points. C'est pourquoi il est nécessaire de supposer l'hypothèse de décidabilité `H2` suivante :

```
Hypothesis H2 : forall u : E, forall eps : posreal,
          decidable (exists u : E, ¬¬phi u ∧ f u <> 0). (* f : E' *)
```

Il est néanmoins possible d'énoncer une version intuitionniste du théorème de Riesz–Fréchet, assurant la « non-non-existence » d'un élément plutôt que son existence.

3. En pratique, c'est le cas des sous-espaces de dimension finie.

8.6.4 Conclusion

La formalisation présentée regroupe un nombre important de définitions et résultats fondamentaux d'analyse fonctionnelle. La nature des concepts formalisés est variée, allant des espaces préhilbertiens et hilbertiens à la complétude des sous-espaces de dimension finie des espaces de Hilbert, en passant par les espaces de fonctions bilinéaires et linéaires continues et par les théorèmes de Riesz–Fréchet et Lax–Milgram.

La preuve Coq du théorème de Lax–Milgram achève la première étape cruciale de la formalisation. Elle est constituée d'environ 7000 lignes de code Coq et est comparable en taille avec la preuve papier détaillée que nous avons suivie (en termes de lignes de code source LaTeX, *i.e.* ≃ 50 pages) [50]. Un constat similaire a été établi par Boldo *et al.* pour des démonstrations liées à la résolution numérique de l'équation des ondes [25]. Ces facilités sont les bénéfices de l'utilisation de la bibliothèque Coquelicot, qui contient l'essentiel des notions basiques d'analyse réelle [35]. La hiérarchie algébrique a notamment été utilisée et étendue : si l'usage des structures canoniques est parfois technique, il permet d'alléger les notations et d'obtenir des énoncés de théorèmes très proches de ceux de la littérature mathématique.

En revanche, la preuve de complétude des sous-espaces de dimension finie a été plus ardue que ce que nous avions envisagé. Si la preuve papier est relativement concise [50] (environ 3 pages), la preuve formelle est quant à elle constituée de près de 1500 lignes de code Coq. Cela s'explique par le caractère hautement topologique des démonstrations. Coquelicot utilise en effet le formalisme des filtres, qui rend la structure des raisonnements inhabituelle par rapport aux preuves papier sur lesquelles nous nous basons.

Une autre difficulté rencontrée dans l'ensemble de la formalisation est l'usage fastidieux des sous-structures, totalement occulté dans les preuves papier. De même, les numériciens éludent généralement les questions liées à la décidabilité, *e.g.* dans la preuve papier de Clément et Martin qui utilise des arguments non constructifs [50].

Chapitre 9

Conclusion et perspectives

Dans la section 9.1, nous proposons un bilan des contributions de la thèse et discutons certaines des difficultés rencontrées. Les sections 9.2 et 9.3 proposent des perspectives respectivement pour les travaux de la partie II et de la partie III du manuscrit.

9.1 Bilan des contributions et difficultés rencontrées

Formalisation d'un algorithme en arrondi correct pour le calcul de moyenne de nombres flottants décimaux

Une première contribution est l'exhibition et la formalisation d'un algorithme fournissant l'arrondi correct de la moyenne de deux nombres flottants décimaux, en tenant compte des dépassements de capacité inférieurs. Nous en tirons plusieurs conclusions.

Ce travail a montré à quel point la propriété d'arrondi correct pouvait être difficile à obtenir et démontrer, même pour des algorithmes relativement simples. Dans le cas du calcul de moyenne, nous avons constaté que l'arrondi correct d'un programme en base 2 n'était pas forcément transposable en base 10 sans modifier substantiellement l'implémentation. Nous avons exhibé un nouvel algorithme et avons montré qu'il vérifiait la propriété d'arrondi correct pour toute base paire.

La preuve de correction de l'algorithme est difficile à vérifier manuellement. Elle repose en effet sur des propriétés fines d'arithmétique à virgule flottante et sur plusieurs disjonctions de cas selon la valeur des entrées du programme. Coq, et plus particulièrement la bibliothèque Flocq [37, 38] d'arithmétique des ordinateurs, se sont révélés comme des alliés. En effet, la majorité des propriétés fondamentales nécessaires à l'établissement de la preuve de correction étaient déjà présentes dans Flocq. Nous avons utilisé Coq comme garde-fou contre d'éventuels raisonnements hâtifs et erronés, notamment pour la gestion des dépassements graduels inférieurs, qui rend la preuve plus difficile et davantage sujette aux erreurs de raisonnement. La preuve formelle, qui est relativement concise, représente environ 700 lignes de code Coq.

Analyse et formalisation d'erreurs d'arrondi de méthodes de Runge-Kutta

La seconde contribution est la mise au point et la formalisation d'une méthodologie générique pour borner les erreurs d'arrondi de l'implémentation de méthodes de résolution d'équations différentielles ordinaires en arithmétique à virgule flottante.

L'approche proposée permet de borner l'accumulation des erreurs d'arrondi induites par l'application de méthodes de Runge-Kutta à des systèmes linéaires. Nous nous sommes dans un premier temps restreints aux systèmes linéaires unidimensionnels, *i.e.* scalaires, puis avons étendu la méthodologie aux systèmes linéaires multidimensionnels, *i.e.* matriciels. La méthodologie est applicable à toute forme d'implémentation et est basée sur une distinction entre les erreurs locales et globales. L'essentiel de la difficulté réside dans la construction de la borne sur l'erreur locale, qui est obtenue incrémentalement suivant la forme de l'implémentation choisie. En particulier, c'est au cours de cette phase de l'analyse que sont utilisées les propriétés des nombres flottants et des opérations matricielles. À partir des bornes d'erreurs locales, une borne sur l'erreur globale est obtenue par application d'un résultat générique, qui montre que l'accumulation des erreurs est maîtrisée sous certaines conditions de stabilité.

Une particularité de ce travail est la gestion rigoureuse des dépassements de capacité (inférieurs et supérieurs dans le cas scalaire, inférieurs uniquement dans le cas matriciel), souvent négligés dans les travaux d'analyse d'erreur.

De plus, nous tirons parti de récents travaux de Jeannerod et Rump [119], qui proposent des bornes plus fines et plus lisibles sur l'erreur d'arrondi des opérations flottantes élémentaires, bornes que nous propageons dans la plupart de nos résultats.

L'analyse des erreurs d'arrondi dans le cas matriciel a été formalisée dans l'assistant de preuves Coq, *via* une combinaison de la bibliothèque Flocq [37, 38] et des matrices de la bibliothèque Mathematical Components [144]. Cette formalisation inclut la démonstration de bornes sur les erreurs d'arrondi d'opérations matricielles, la définition des méthodes de Runge-Kutta explicites et de leur implémentation, les lemmes génériques permettant de reconstruire les bornes d'erreurs locales et globales ainsi que l'application de ces résultats aux méthodes d'Euler et de Runge-Kutta d'ordre 2. La formalisation proposée comporte environ 4200 lignes de code Coq.

Dans les démonstrations formelles, nous avons également utilisé des tactiques automatiques existantes. Pour borner les erreurs locales des méthodes classiques, nous avons par exemple utilisé la tactique `interval` [150] pour majorer les termes d'ordre supérieurs (en u^2, u^3, etc). De plus, les énoncés de certains résultats sont difficiles à établir, *e.g.* les lemmes permettant d'obtenir la borne d'erreur locale (lemme 6.8 et corollaire 6.9) et leur application. Plutôt que d'énoncer et démontrer ces résultats sur papier, au risque de faire des erreurs, puis de les démontrer formellement, nous avons directement utilisé Coq comme aide à la reconstruction du résultat. Nous avons utilisé le mécanisme Coq des variables existentielles (`evars`), *i.e.* des termes de la forme `?X` dont la détermination pourra être remise à plus tard, *e.g.* en accumulant des contraintes par application de théorèmes dans les buts où ils apparaissent. Nous avons par exemple utilisé les tactiques `eapply` et `etransitivity` qui introduisent des `evars`.

Formalisation de résultats d'analyse fonctionnelle

La méthode des éléments finis constitue l'une des méthodes numériques les plus puissantes pour résoudre les équations aux dérivées partielles. La convergence de la méthode repose sur un socle conséquent de fondements mathématiques. La troisième contribution de ma thèse est la formalisation d'une partie de ces fondements.

La formalisation proposée repose sur la bibliothèque Coquelicot [35, 134]. Nous avons étendu la hiérarchie algébrique aux espaces préhilbertiens et hilbertiens et avons formalisé des concepts et résultats géométriques propres à ces classes d'espaces. Le développement inclut par ailleurs la formalisation complète du théorème de Lax–Milgram, un résultat fondamental d'analyse fonctionnelle qui pourra être utilisé pour démontrer la convergence de la méthode des éléments finis. Enfin, pour caractériser les maillages d'éléments finis, nous avons formalisé la notion de sous-espace de dimension finie des espaces de Hilbert et avons démontré qu'ils constituent des sous-modules complets.

Dans la littérature mathématique, la démonstration de ces résultats est souvent écrite de manière concise, les détails techniques étant cachés. Cela rend particulièrement difficile leur formalisation en Coq. Nous avons donc mené ce travail en collaboration avec deux numériciens, qui ont proposé une preuve papier détaillée du théorème de Lax–Milgram, conçue spécifiquement dans l'objectif d'une formalisation [50]. Cette démonstration papier a globalement été suivie, mais a fait l'objet de modifications au cours de la formalisation, au gré de discussions entre numériciens et utilisateurs de Coq.

Nous avons néanmoins été confronté à des difficultés inhérentes à la formalisation, qui n'apparaissent pas dans les preuves papier. Par exemple, alors que les sous-structures sont complètement transparentes dans la démonstration de Clément et Martin [50], leur gestion en Coq s'avère particulièrement difficile. Les raisonnements topologiques constituent une seconde difficulté majeure. En effet, la démonstration papier que nous avons suivie utilise le formalisme des suites, intuitif pour le lecteur, alors que Coquelicot repose sur le formalisme plus général des filtres. Nous avons donc dû adapter les définitions et les démonstrations topologiques au formalisme des filtres. Une troisième difficulté réside dans le choix des axiomes utilisés. Coq repose sur la logique intuitionniste alors que les preuves papier d'analyse fonctionnelle reposent sur des fondements classiques. Nous avons donc fait le choix d'identifier précisément les prédicats sur lesquels une propriété de décidabilité était nécessaire. Nous utilisons les axiomes `ProofIrrelevance` et `FunctionalExtensionality`, qui facilitent la formalisation des résultats et sont requis dans des formalisations récentes d'analyse, *e.g.* la bibliothèque MathComp-Analysis [1].

La formalisation des espaces de Hilbert et du théorème de Lax–Milgram constituent environ 7000 lignes de code Coq, ce qui est comparable en magnitude avec la preuve papier de Clément et Martin, longue d'environ 50 pages. En revanche, la preuve de fermeture des sous-espaces de dimension finie fait à elle seule 1500 lignes de code Coq pour seulement 3 pages de preuves papier. Cela s'explique par la prépondérance de raisonnements topologiques nécessitant l'utilisation de transformateurs ou de projecteurs de filtres (voir section 8.5) et la manipulation de sous-structures, qui n'est pas aussi naturelle que dans les preuves papier (voir section 7.2).

9.2 Perspectives sur les méthodes de Runge-Kutta

Dans cette section, nous présentons quelques perspectives possibles pour les résultats de la partie II du manuscrit.

9.2.1 Comparaisons entre les erreurs de méthode et d'arrondi

Nous avons proposé une méthodologie générique permettant de borner l'erreur d'arrondi de méthodes de Runge-Kutta. Cela a permis de montrer, au moins dans le cas de couples systèmes-méthodes stables, que les erreurs d'arrondi ne s'accumulent pas de manière catastrophique. Une perspective intéressante serait de déterminer la contribution des erreurs de méthode par rapport aux erreurs d'arrondi.

Dans la section 5.7, nous avons présenté, pour la méthode RK2 appliquée à un système linéaire scalaire, des résultats expérimentaux permettant de comparer la borne d'erreur d'arrondi aux erreurs de méthode et d'arrondi effectivement commises. Cela a permis de montrer que la borne obtenue pour cet exemple jouet, bien que relativement pessimiste, garantissait que l'erreur de méthode domine largement l'erreur d'arrondi. Nous souhaiterions mener le même type d'expérimentation pour la résolution numérique d'un système linéaire matriciel caractérisant un phénomène physique concret. Nous souhaiterions plus précisément éprouver notre analyse sur des équations de Bateman [12] qui caractérisent des phénomènes de décroissance radioactive. Il s'agit en effet d'équations différentielles matricielles non raides, utilisées dans des applications industrielles et généralement résolues *via* des méthodes de Runge-Kutta.

L'approche empirique proposée ci-dessus ne permet pas de comparer les erreurs de méthode et d'arrondi dans le cas général. Une possibilité est de borner non seulement l'erreur d'arrondi, mais aussi l'erreur de méthode. Il s'agit de l'approche adoptée par Fousse [79, 78, 77] pour les méthodes de calcul numérique d'intégrales, qui borne l'erreur totale, *i.e.* la somme des erreurs de méthode et des erreurs d'arrondi.

En général, les numériciens ne donnent qu'une estimation de l'erreur de méthode [45]. En l'occurrence, l'utilisation d'une méthode de Runge-Kutta d'ordre p assure que l'erreur de méthode est de l'ordre de $O(h^{p+1})$. En revanche, l'estimation de la constante associée au grand O est difficile à exhiber dans le cas général [46].

9.2.2 Généralisation à d'autres classes de méthodes

Méthodes implicites :

Une perspective de travail envisagée est l'adaptation de l'analyse d'erreurs aux méthodes de Runge-Kutta implicites, *e.g.* les méthodes d'Euler implicite et du point-milieu implicite [59] (voir section 2.3.3), particulièrement adaptées à la résolution de systèmes *raides*. Une équation différentielle est dite *raide* lorsque sa résolution par des méthodes numériques explicites est difficile, *i.e.* lorsque les méthodes explicites sont instables sur cette équation, même lorsque le pas d'intégration choisi est très petit. Les équations

raides sont particulièrement répandues en chimie, notamment pour modéliser des réactions chimiques dont l'évolution est brutale [99].

L'implémentation des méthodes implicites est plus complexe et nécessite généralement la résolution d'une équation algébrique. Par exemple, la méthode d'Euler implicite appliquée à l'équation $y' = f(y,t)$ est caractérisée par la relation itérative :

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}),$$

qui fait apparaître le terme y_{n+1} des deux côtés de l'équation.

Des simplifications sont cependant possibles dans le cas des équations linéaires de la forme $y' = Ay$, avec $y \in \mathbb{R}^n$ et $A \in \mathbb{R}^{n\times n}$. La relation itérative associée à la méthode d'Euler est alors :

$$y_{n+1} = y_n + hAy_{n+1},$$

qui peut être transformée en $y_{n+1} = (I - hA)^{-1}y_n$.

À partir de cette caractérisation, nous souhaitons adapter la méthodologie adoptée dans le chapitre 6 pour borner les erreurs d'arrondi. Une possibilité est de borner l'erreur d'arrondi commise au cours de l'inversion de la matrice $(I-hA)^{-1}$, ce qui pose plusieurs difficultés :

- Cela dépend de la méthode d'inversion utilisée, *e.g.* par une inversion par blocs ou *via* la décomposition LU de la matrice. Higham a proposé des bornes sur les erreurs d'arrondi associées à ces méthodes d'inversion [107, §14]. Cependant, les bornes proposées font intervenir des quantités qui ne sont pas forcément connues *a priori*, *e.g.* la norme des matrices L et U de la décomposition LU.
- De plus, la matrice $I-hA$ n'est en général pas calculée de manière exacte, si bien que la matrice à inverser est de la forme $I \ominus h \otimes \widetilde{A}$. Il faudra donc analyser la sensibilité de la méthode d'inversion aux perturbations de la matrice, *i.e.* borner l'erreur $|||(I \ominus h \otimes \widetilde{A})^{-1} - (I - hA)^{-1}|||$.

Méthodes multi-pas :

Une autre extension possible est l'adaptation de la méthodologie aux méthodes multi-pas, *e.g.* les méthodes d'Adams-Moulton et d'Adams-Bashforth [59]. Un aspect intéressant de ces méthodes est le fait que la relation de récurrence qui les caractérise fasse apparaître une alternance de signes. Par exemple, la méthode d'Adams-Bashforth à trois pas est caractérisée par :

$$y_{n+3} = y_{n+2} + h\left(\frac{23}{12}f(y_{n+2},t_{n+2}) - \frac{16}{12}f(y_{n+1},t_{n+1}) + \frac{5}{12}f(y_n,t_n)\right).$$

Ainsi, dans le cas des systèmes linéaires de la forme $y' = Ay$, nous avons :

$$y_{n+3} = y_{n+2} + h\left(\frac{23}{12}Ay_{n+2} - \frac{16}{12}Ay_{n+1} + \frac{5}{12}Ay_n\right).$$

En considérant les erreurs locales et globales comme étant signées, nous obtenons :

$$E_{n+3} = \varepsilon_{n+3} + h\left(\frac{23}{12}E_{n+2} - \frac{16}{12}E_{n+1} + \frac{5}{12}E_n\right).$$

Il serait intéressant de déterminer si l'alternance de signes permet d'exhiber des compensations d'erreurs, *e.g.* en donnant une expression analytique des erreurs globales, dans l'esprit des travaux de Boldo pour l'équation des ondes [23] (voir section 3.2.2).

9.2.3 Généralisation aux systèmes non-linéaires

Une autre perspective est l'adaptation d'une analyse d'erreurs à grains fins dans le cas de systèmes non-linéaires quelconques. Certains auteurs, *e.g.* Spijker [192] ou Demailly [59], ont exhibé des bornes génériques sur les erreurs d'arrondi en tirant parti de la stabilité des méthodes. Ces bornes s'avèrent cependant grossières dans la plupart des cas et n'utilisent pas les propriétés fines du format d'arithmétique (voir section 3.1.3).

Dans la partie II de ce manuscrit, afin d'obtenir des bornes plus fines, nous avons décomposé l'analyse des erreurs d'arrondi jusqu'au niveau des opérations flottantes élémentaires. Cette approche a été rendu possible par la restriction aux calculs matriciels, qui sont uniquement composés de sommations et de multiplications. Une telle approche n'est pas adaptée à des fonctions non-linéaires quelconques.

Nous avons néanmoins identifié des pistes qui pourraient permettre d'étudier les erreurs d'arrondi de certains systèmes non-linéaires. Nous pourrions utiliser les bornes d'erreurs de certaines fonctions élémentaires, *e.g.* les fonctions trigonométriques, que les fabricants de processeurs fournissent. Ces bornes ne sont cependant pas vérifiables et pourraient s'avérer erronées. De plus, il est difficile d'évaluer la sensibilité de ces fonctions à d'éventuelles perturbations sur les paramètres en entrée. Une autre possibilité est de considérer la linéarisation d'un système non-linéaire $y' = f(y,t)$ *via* le calcul de la matrice jacobienne de f et d'étudier les erreurs d'arrondi sur le système linéaire obtenu. Dans ce cas, en toute rigueur, il faudrait que nous soyons capable de borner les erreurs associées au processus de linéarisation.

9.2.4 Vers une meilleure utilisation des `evars`

Notre utilisation du mécanisme des variables existentielles de Coq comme aide à la reconstruction de certains résultats est relativement rudimentaire. Nous pourrions envisager d'utiliser ou d'adapter de récents travaux qui permettent de faciliter ce type de chemins de preuve.

C'est par exemple le cas de la bibliothèque Procrastination développée par Guéneau [93] et utilisée dans des preuves de complexité asymptotique d'algorithmes [92] et de la tactique automatique `near` proposée par Affeldt, Cohen et Rouhling [52, 2] pour des preuves sur les filtres (au sens topologique), *e.g.* des preuves de convergence.

9.2.5 Preuve de programmes

Nous pourrions envisager la vérification formelle des propriétés flottantes de programmes existants (écrits en C par exemple) qui implémentent une méthode de Runge-Kutta. Nous pourrions utiliser la méthodologie et le langage d'annotations proposés par Boldo et Filliâtre [32] pour la vérification de programmes flottants dans Frama-C. Notre idée serait d'ensuite utiliser notre formalisation Coq en arrière-plan en interprétant le programme comme une application de notre méthodologie d'analyse d'erreur. Dans le même esprit que pour l'équation des ondes [25], nous pourrions envisager de vérifier à la fois les propriétés mathématiques et flottantes du programme.

Une alternative possible serait d'étudier s'il est possible d'utiliser le mécanisme d'extraction de Coq [136] sur notre développement afin d'extraire d'une part une implémentation de la méthode numérique en OCaml et d'autre part une borne d'erreur garantie.

9.3 Perspectives sur la méthode des éléments finis

Dans cette section, nous présentons d'éventuelles perspectives pour les travaux que nous présentons dans la partie III de ce manuscrit.

9.3.1 Intégration de Lebesgue et espaces de Sobolev

Le théorème de Lax–Milgram que nous avons démontré formellement dans le chapitre 8 s'applique à tout espace de Hilbert. Pour établir la convergence de la méthode des éléments finis, il faut instancier ce résultat à des espaces fonctionnels particuliers, à savoir des espaces de Sobolev construits à partir de l'espace L^2 des fonctions de carré intégrable (au sens de la théorie de l'intégration de Lebesgue) [49].

Cette nouvelle étape de formalisation présente plusieurs difficultés :

- L'espace L^2 repose sur la théorie de la mesure et de l'intégration de Lebesgue, dont il n'existe pas de formalisation aboutie en Coq. Avec Boldo, Clément, Martin et Mayero, nous sommes en phase finale de formalisation de l'intégrale de Lebesgue et de la démonstration de certains résultats fondamentaux comme les théorèmes de Beppo Levi (convergence monotone), de Fatou–Lebesgue et de convergence dominée [178]. La démonstration de ces résultats s'avère difficile car elle repose sur un socle important de concepts, *e.g.* les tribus, la mesurabilité et l'intégrabilité de fonctions, la notion de mesure ou les fonctions étagées.
- La formalisation de l'espace L^2 et la démonstration de sa complétude posent également des difficultés techniques, notamment du fait que L^2 soit construit comme un espace quotient, dont la relation de quotientage est l'égalité presque partout entre fonctions. La preuve de complétude nécessite par ailleurs une fonction limite sur les filtres (voir section 7.1.2), pouvant s'avérer difficile à exhiber.

9.3.2 De l'analyse fonctionnelle à la méthode des éléments finis

Théorie des maillages :

Dans le chapitre 8, nous avons démontré que les sous-espaces de dimension finie d'un espace de Hilbert E étaient des sous-modules complets de E, ce qui a permis de leur appliquer le théorème de Lax–Milgram. Pour la méthode des éléments finis, les sous-espaces de dimension finie à considérer sont des maillages de volumes irréguliers. Une perspective de travail à moyen terme serait une formalisation de ces maillages qui soit compatible avec la théorie sous-jacente des espaces de Hilbert et des sous-espaces de dimension finie. L'ensemble de la formalisation pourrait ensuite servir de base à une démonstration de la convergence de la méthode des éléments finis pour un maillage particulier ou pour une classe donnée de maillages.

Certification d'une bibliothèque C++ d'éléments finis :

La formalisation d'analyse fonctionnelle présentée dans la partie III de ce document s'inscrit dans un projet visant à certifier tout ou partie de FELiScE [1], une vaste bibliothèque C++ d'éléments finis constituée de plus de 100 000 lignes de code. En particulier, nous souhaiterions établir la correction des programmes implémentés en utilisant (entre autres) la propriété de convergence de la méthode des éléments finis.

D'un point de vue pratique, une possibilité serait d'utiliser Frama-Clang [169], un *plug-in* Frama-C qui permet de démontrer des programmes C++ et dont le langage d'annotations, ACSL++, est très similaire au langage ACSL utilisé dans Frama-C pour annoter les programmes écrits en C.

Analyse des erreurs d'arrondi de la méthode des éléments finis :

Nous nous sommes jusqu'ici principalement intéressé aux propriétés mathématiques de la méthode des éléments finis. Les programmes manipulant la méthode des éléments finis sont généralement complexes et impliquent de nombreux calculs. Nous pourrions envisager, comme pour les méthodes de Runge-Kutta, d'analyser et de formaliser les erreurs d'arrondi associées à ces implémentations. Néanmoins, dans la méthode des éléments finis, contrairement aux méthodes de Runge-Kutta, les potentielles sources d'erreurs d'arrondi sont très nombreuses. Elles peuvent par exemple provenir de la génération du maillage, des coefficients apparaissant dans l'équation aux dérivées partielles ou de sa linéarisation sur le maillage.

Concernant la vérification déductive de propriétés flottantes de programmes, *e.g.* pour la bibliothèque FELiScE, nous estimons qu'il est possible d'utiliser la méthodologie et l'ensemble d'annotations dédiées proposées par Boldo et Filliâtre [32] et de l'adapter à Frama-Clang et ACSL++.

1. Finite Elements for Life Sciences and Engineering
`https://gforge.inria.fr/projects/felisce/`

Bibliographie

[1] Affeldt, Reynald, Cyril Cohen, Assia Mahboubi, Damien Rouhling et Pierre Yves Strub: *Classical analysis with Coq*. Dans *10th Coq Workshop*, 2018.

[2] Affeldt, Reynald, Cyril Cohen et Damien Rouhling: *Formalization Techniques for Asymptotic Reasoning in Classical Analysis*. Journal of Formalized Reasoning, octobre 2018. `https://hal.inria.fr/hal-01719918`.

[3] Afshar, Sanaz Khan, Vincent Aravantinos, Osman Hasan et Sofiène Tahar: *Formalization of Complex Vectors in Higher-Order Logic*. Dans Watt, Stephen M., James H. Davenport, Alan P. Sexton, Petr Sojka et Josef Urban (rédacteurs): *Intelligent Computer Mathematics*, pages 123–137, Cham, 2014. Springer International Publishing.

[4] Alt, René et Jean Vignes: *Validation of results of collocation methods for ODEs with the CADNA library*. Applied Numerical Mathematics, 21(2):119–139, 1996.

[5] Antoñana, Mikel, Joseba Makazaga et Ander Murua: *Reducing and monitoring round-off error propagation for symplectic implicit Runge-Kutta schemes*. Numerical Algorithms, 76(4):861–880, 2017.

[6] Arnold, Martin, Bernhard Burgermeister, Claus Führer, Gerhard Hippmann et Georg Rill: *Numerical methods in vehicle system dynamics: state of the art and current developments*. Vehicle System Dynamics, 49(7):1159–1207, 2011.

[7] Assaf, Ali: *A calculus of constructions with explicit subtyping*. Dans Herbelin, Hugo, Pierre Letouzey et Matthieu Sozeau (rédacteurs): *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, tome 39 de *Leibniz International Proceedings in Informatics*, Institut Henri Poincaré, Paris, France, mai 2014. `https://hal.archives-ouvertes.fr/hal-01097401`.

[8] Baier, Christel et Joost Pieter Katoen: *Principles of model checking*. MIT press, 2008.

[9] Bailey, David, Peter Borwein et Simon Plouffe: *On the rapid computation of various polylogarithmic constants*. Mathematics of Computation of the American Mathematical Society, 66(218):903–913, 1997.

[10] Baker Jr, George A. et Peter Graves-Morris: *Padé approximants*, tome 59. Cambridge University Press, 1996.

[11] Barrett, Geoff: *Formal methods applied to a floating-point number system*. IEEE Transactions on Software Engineering, 15(5):611–621, 1989.

[12] Bateman, Harry: *The solution of a system of differential equations occurring in the theory of radio-active transformations*. Proceedings of the Cambridge Philosophical Society, 1908, 15:423–427, 1908.

[13] Bauer, Andrej, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau et Bas Spitters: *The HoTT library: a formalization of homotopy type theory in Coq*. Dans *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 164–172. ACM, 2017.

[14] Bauer, Andrej et Davorin Lešnik: *Metric spaces in synthetic topology*. Annals of Pure and Applied Logic, 163(2):87–100, 2012.

[15] Bauer, Gertrud et Makarius Wenzel: *Computer-Assisted Mathematics at Work*. Dans *International Workshop on Types for Proofs and Programs*, pages 61–76. Springer, 1999.

[16] Ben-Ari, Mordechai: *The bug that destroyed a rocket*. ACM Special Interest Group on Computer Science Education (SIGCSE) Bulletin, 33(2):58–59, 2001.

[17] Bertot, Yves et Pierre Castéran: *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An European Association for Theoretical Computer Science Series. Springer, 2004. https://doi.org/10.1007/978-3-662-07964-5.

[18] Bertot, Yves, Georges Gonthier, Sidi Ould Biha et Ioana Paşca: *Canonical Big Operators*. Dans *Theorem Proving in Higher Order Logics*, tome 5170/2008 de *Lecture Notes in Computer Science*, Montreal, Canada, août 2008. https://hal.inria.fr/inria-00331193.

[19] Bertot, Yves, Laurence Rideau et Laurent Théry: *Distant decimals of π*. Journal of Automated Reasoning, pages 1–45, 2017. https://hal.inria.fr/hal-01582524.

[20] Berz, Martin et Kyoko Makino: *Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models*. Reliable Computing, 4(4):361–369, 1998.

[21] Bodenheimer, Peter, Gregory P. Laughlin, Michal Rozyczka, Tomasz Plewa et Harold W. Yorke: *Numerical Methods in Astrophysics: An Introduction*. Series in Astronomy and Astrophysics. CRC Press, 2006.

[22] Bohrer, Brandon, Vincent Rahli, Ivana Vukotic, Marcus Völp et André Platzer: *Formally verified differential dynamic logic*. Dans *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 208–221. ACM, 2017.

[23] Boldo, Sylvie: *Floats & Ropes: a case study for formal numerical program verification*. Dans *36th International Colloquium on Automata, Languages and Programming*, tome 5556 de *Lecture Notes in Computer Science - ARCoSS*, pages 91–102, Rhodos, Greece, juillet 2009. Springer. http://fost.saclay.inria.fr/files/icalp2009trackb_submission_48.pdf.

[24] Boldo, Sylvie: *Formal Verification of Programs Computing the Floating-Point Average*. Dans Butler, Michael, Sylvain Conchon et Fatiha Zaïdi (rédacteurs): *17th International Conference on Formal Engineering Methods*, tome 9407 de *Lecture Notes in Computer Science*, pages 17–32, Paris, France, novembre 2015. Springer International Publishing. https://hal.inria.fr/hal-01174892.

[25] Boldo, Sylvie, François Clément, Jean Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond et Pierre Weis: *Wave Equation Numerical Resolution: a Comprehensive Mechanized Proof of a C Program*. Journal of Automated Reasoning, 50(4):423–456, avril 2013. http://hal.inria.fr/hal-00649240/en/.

[26] Boldo, Sylvie, François Clément, Florian Faissole, Vincent Martin et Micaela Mayero: *A Coq Formal Proof of the Lax–Milgram theorem*. Dans *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 79–89, Paris, France, janvier 2017. ACM. https://hal.inria.fr/hal-01391578.

[27] Boldo, Sylvie, François Clément, Florian Faissole, Vincent Martin et Micaela Mayero: *Preuve formelle du théorème de Lax–Milgram*. Dans *16èmes journées Approches Formelles dans l'Assistance au Développement de Logiciels*, Montpellier, France, juin 2017. https://hal.archives-ouvertes.fr/hal-01581807.

[28] Boldo, Sylvie, François Clément, Jean Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond et Pierre Weis: *Formal Proof of a Wave Equation Resolution Scheme: the Method Error*. Dans Kaufmann, Matt et Lawrence C. Paulson (rédacteurs): *Proceedings of the first Interactive Theorem Proving Conference (ITP)*, tome 6172 de *Lecture Notes in Computer Science*, pages 147–162, Edinburgh, Scotland, juillet 2010. Springer. http://hal.inria.fr/inria-00450789/en/, (merge of TPHOL and ACL2).

[29] Boldo, Sylvie, Florian Faissole et Alexandre Chapoutot: *Round-off Error Analysis of Explicit One-Step Numerical Integration Methods*. Dans *24th IEEE Symposium on Computer Arithmetic*, London, United Kingdom, juillet 2017. https://hal.archives-ouvertes.fr/hal-01581794.

[30] Boldo, Sylvie, Florian Faissole et Alexandre Chapoutot: *Round-off error and exceptional behavior analysis of explicit Runge-Kutta methods*. IEEE Transactions on Computers, 2019. https://hal.archives-ouvertes.fr/hal-01883843.

[31] Boldo, Sylvie, Florian Faissole et Vincent Tourneur: *A Formally-Proved Algorithm to Compute the Correct Average of Decimal Floating-Point Numbers*. Dans *25th IEEE Symposium on Computer Arithmetic*, Amherst, MA, United States, juin 2018. https://hal.inria.fr/hal-01772272.

[32] Boldo, Sylvie et Jean Christophe Filliâtre: *Formal Verification of Floating-Point Programs*. Dans Kornerup, Peter et Jean Michel Muller (rédacteurs): *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 187–194, Montpellier, France, juin 2007. http://www.lri.fr/~filliatr/ftp/publis/caduceus-floats.pdf.

[33] Boldo, Sylvie, Stef Graillat et Jean Michel Muller: *On the robustness of the 2Sum and Fast2Sum algorithms*. ACM Transactions on Mathematical Software, 44(1), juillet 2017. https://hal-ens-lyon.archives-ouvertes.fr/ensl-01310023.

[34] Boldo, Sylvie, Mioara Joldes, Jean Michel Muller et Valentina Popescu: *Formal Verification of a Floating-Point Expansion Renormalization Algorithm*. Dans *Proceedings of the 8th International Conference on Interactive Theorem Proving*, Brasilia, Brazil, septembre 2017. https://hal.archives-ouvertes.fr/hal-01512417.

[35] Boldo, Sylvie, Catherine Lelay et Guillaume Melquiond: *Coquelicot: A User-Friendly Library of Real Analysis for Coq*. Mathematics in Computer Science, 9(1):41–62, 2015, ISSN 1661-8270. http://dx.doi.org/10.1007/s11786-014-0181-1.

[36] Boldo, Sylvie, Catherine Lelay et Guillaume Melquiond: *Formalization of Real Analysis: A Survey of Proof Assistants and Libraries*. Mathematical Structures in Computer Science, 26(7):1196–1233, 2016. http://hal.inria.fr/hal-00806920.

[37] Boldo, Sylvie et Guillaume Melquiond: *Flocq: A Unified Library for Proving Floating-point Algorithms in Coq*. Dans Antelo, Elisardo, David Hough et Paolo Ienne (rédacteurs): *20th IEEE Symposium on Computer Arithmetic*, pages 243–252, Tübingen, Germany, 2011.

[38] Boldo, Sylvie et Guillaume Melquiond: *Computer Arithmetic and Formal Proofs*. ISTE Press - Elsevier, décembre 2017.

[39] Bouissou, Olivier, Alexandre Chapoutot et Adel Djoudi: *Enclosing Temporal Evolution of Dynamical Systems Using Numerical Methods*. Dans *NASA Formal Methods*, numéro 7871 dans *Lecture Notes in Computer Science*, pages 108–123. Springer, 2013.

[40] Bouissou, Olivier et Matthieu Martel: *GRKLib: a Guaranteed Runge-Kutta Library*. Dans *International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*. IEEE, 2006.

[41] Bourbaki, Nicolas: *Topologie générale: Chapitres 1 à 4*. Bourbaki, Nicolas. Springer Berlin Heidelberg, 1971.

[42] Brézis, Haïm, Philippe G. Ciarlet et Jacques Louis Lions: *Analyse fonctionnelle: théorie et applications*. Collection Mathématiques appliquées pour la maîtrise. Dunod, 1999. https://books.google.fr/books?id=aykKPQAACAAJ.

[43] Brouwer, Dirk: *On the accumulation of errors in numerical integration*. The Astronomical Journal, 46:149–153, 1937.

[44] Brunel, Aloïs: *Non-constructive complex analysis in Coq*. Dans *18th International Workshop on Types for Proofs and Programs*, pages 1–15, Bergen, Norvège, septembre 2011. http://dx.doi.org/10.4230/LIPIcs.TYPES.2011.1.

[45] Butcher, John Charles et Nicolette Goodwin: *Numerical methods for ordinary differential equations*, tome 2. Wiley Online Library, 2008.

[46] Butcher, John Charles et P. B. Johnston: *Estimating local truncation errors for Runge-Kutta methods*. Journal of Computational and Applied Mathematics, 45(1-2):203–212, 1993.

[47] Carreño, Victor A et Paul S Miner: *Specification of the IEEE-854 floating-point standard in HOL and PVS*. High Order Logic Theorem Proving and Its Applications, 16, 1995.

[48] Cartan, Henri: *Théorie des filtres*. Comptes Rendus de l'Académie des Sciences, Paris, 205:595–598, 1937.

[49] Ciarlet, Philippe G.: *The finite element method for elliptic problems*, tome 40 de *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2002. `http://dx.doi.org/10.1137/1.9780898719208`, Reprint of the 1978 original [North-Holland, Amsterdam; MR0520174 (58 #25001)].

[50] Clément, François et Vincent Martin: *The Lax-Milgram Theorem. A detailed proof to be formalized in Coq*. Research Report RR-8934, Inria Paris, juillet 2016. `https://hal.inria.fr/hal-01344090`.

[51] Cody, William J., Jerome T. Coonen, David M. Gay, Kenton Hanson, David Hough, William Kahan, Richard Karpinski, John Palmer, Frederic N. Ris et David Stevenson: *A proposed radix-and word-length-independent standard for floating-point arithmetic*. IEEE Micro, (4):86–100, 1984.

[52] Cohen, Cyril: *Formalized algebraic numbers: construction and first-order theory.* Thèse de doctorat, Ecole Polytechnique X, 2012.

[53] Cousot, Patrick et Radhia Cousot: *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*. Dans *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[54] Cowlishaw, Michael F: *Decimal floating-point: Algorism for computers*. Dans *16th IEEE Symposium on Computer Arithmetic*, pages 104–111. IEEE, 2003.

[55] Cruz-Filipe, Luís, Herman Geuvers et Freek Wiedijk: *C-CoRN, the constructive Coq repository at Nijmegen*. Dans *International Conference on Mathematical Knowledge Management*, pages 88–103. Springer, 2004.

[56] Daramy, Catherine, David Defour, Florent de Dinechin et Jean Michel Muller: *CR-LIBM: a correctly rounded elementary function library*. Dans *Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, tome 5205, pages 458–464. International Society for Optics and Photonics, 2003.

[57] Daumas, Marc et Guillaume Melquiond: *Certification of bounds on expressions involving rounded operators*. Transactions on Mathematical Software, 37(1):1–20, 2010.

[58] Dekker, Theodorus J.: *A floating-point technique for extending the available precision*. Numerische Mathematik, 18(3):224–242, 1971.

[59] Demailly, Jean Pierre: *Analyse numérique et équations différentielles.* Collection Grenoble sciences. Les Ulis - EDP Sciences, 2016. `https://hal.archives-ouvertes.fr/hal-01675213`.

[60] Desmettre, Olivier: *Coq standard library–RiemannInt.* 2002.

[61] Dinechin, Florent de, Christoph Lauter et Guillaume Melquiond: *Certifying the Floating-point Implementation of an Elementary Function Using Gappa.* IEEE Transactions on Computers, 60(2):242–253, 2011.

[62] Duale, Ali Y, Mark H Decker, Hans-Georg Zipperer, Merav Aharoni et Theodore J Bohizic: *Decimal floating-point in Z9: An implementation and testing perspective.* IBM Journal of Research and Development, 51(1.2):217–227, 2007.

[63] Earn, David J. D.: *Symplectic integration without roundoff error.* Dans *Ergodic Concepts in Stellar Dynamics*, tome 430 de *LNP*, pages 122–130. Springer Berlin Heidelberg, 1994.

[64] Ern, Alexandre et Jean Luc Guermond: *Theory and practice of finite elements*, tome 159 de *Applied Mathematical Sciences.* Springer-Verlag, New York, 2004. `http://dx.doi.org/10.1007/978-1-4757-4355-5`.

[65] Escardó, Martín: *Synthetic topology of data types and classical spaces.* Electronic Notes in Theoretical Computer Science, 87:21–156, 2004.

[66] Euler, Leonhard: *Institutionum calculi integralis, 3 vols.* Petropoli: Impensis Academia Imperialis Scientiarum. In (Euler, OO, Series 1, 11-13), 1768.

[67] Eymard, Robert, Thierry Gallouët et Raphaèle Herbin: *Finite volume methods.* Handbook of Numerical Analysis, 7:713–1018, 2000.

[68] Faissole, Florian: *Formalization and closedness of finite dimensional subspaces.* Dans *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 121–128. IEEE, 2017.

[69] Faissole, Florian: *Définir le fini: deux formalisations d'espaces de dimension finie.* Dans *Journées Francophones des Langages Applicatifs*, pages 167–172, Banyuls-sur-Mer, France, janvier 2018.

[70] Faissole, Florian: *Formalisation en Coq des erreurs d'arrondi de méthodes de Runge-Kutta pour les systèmes matriciels.* Dans *18èmes journées Approches Formelles dans l'Assistance au Développement de Logiciels*, Toulouse, France, juin 2019.

[71] Faissole, Florian et Bas Spitters: *Synthetic topology in HoTT for probabilistic programming.* Dans *The Third International Workshop on Coq for Programming Languages (CoqPL 2017)*, Paris, France, janvier 2017. `https://hal.inria.fr/hal-01405762`.

[72] Faissole, Florian et Bas Spitters: *Un cadre pour la preuve de programmes probabilistes.* Journées Francophones des Langages Applicatifs, pages 137–150, janvier 2018.

[73] Faure, Christèle et Yves Papegay: *Odyssée user's guide version 1.7*. 1998.

[74] Fehlberg, Erwin: *Error propagation in Runge-Kutta type integration formulas*. Rapport technique NASA-TR-R-352, M-530, NASA Marshall Space Flight Center, 1970.

[75] Filliâtre, Jean Christophe: *Deductive software verification*, 2011.

[76] Filliâtre, Jean Christophe et Andrei Paskevich: *Why3—where programs meet provers*. Dans *European Symposium on Programming*, pages 125–128. Springer, 2013.

[77] Fousse, Laurent: *Intégration numérique avec erreur bornée en précision arbitraire*. Thèse de doctorat, 2006.

[78] Fousse, Laurent: *Accurate Multiple-Precision Gauss-Legendre Quadrature*. Dans *18th IEEE Symposium on Computer Arithmetic*, pages 150–160. IEEE Computer Society, 2007.

[79] Fousse, Laurent: *Multiple-Precision Correctly rounded Newton-Cotes quadrature*. Informatique Théorique et Applications, 41(1):103–121, 2007.

[80] Fousse, Laurent, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier et Paul Zimmermann: *MPFR: A multiple-precision binary floating-point library with correct rounding*. ACM Transactions on Mathematical Software (TOMS), 33(2):13, 2007.

[81] Fukushima, Toshio: *Reduction of Round-off Errors in the Extrapolation Methods and its Application to the Integration of Orbital Motion*. Astronomical Journal, 112(3), 1996.

[82] Geuvers, Herman: *Proof assistants: History, ideas and future*. Sadhana, 34(1):3–25, 2009.

[83] Goldberg, David: *What every computer scientist should know about floating-point arithmetic*. ACM Computing Surveys, 23(1):5–48, mars 1991, ISSN 0360-0300. `http://doi.acm.org/10.1145/103162.103163`.

[84] Golub, Gene H. et Charles F. Van Loan: *Matrix Computations*. The Johns Hopkins University Press, 3e édition, 1996.

[85] Gonthier, Georges: *Formal proof–the four-color theorem*. Notices of the American Mathematical Society, 55(11):1382–1393, 2008.

[86] Gonthier, Georges, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Paşca, Laurence Rideau, Alexey Solovyev, Enrico Tassi et Laurent Théry: *A Machine-Checked Proof of the Odd Order Theorem*. Dans Blazy, Sandrine, Christine Paulin et David Pichardie (rédacteurs): *ITP 2013, 4th Conference on Interactive Theorem Proving*, tome 7998 de *Lecture Notes in Computer Science*, pages 163–179, Rennes, France, juillet 2013. Springer. `https://hal.inria.fr/hal-00816699`.

[87] Gonthier, Georges et Assia Mahboubi: *An introduction to small scale reflection in Coq*. Journal of Formalized Reasoning, 3(2):95–152, 2010. `https://hal.inria.fr/inria-00515548`.

[88] Goualard, Frédéric: *How Do You Compute the Midpoint of an Interval?* ACM Transactions on Mathematical Software, 40(2):11:1–11:25, mars 2014, ISSN 0098-3500. `http://doi.acm.org/10.1145/2493882`.

[89] Graillat, Stef, Vincent Lefèvre et Jean Michel Muller: *On the maximum relative error when computing integer powers by iterated multiplications in floating-point arithmetic*. Numerical Algorithms, 70(3):653–667, 2015.

[90] Granas, Andrzej et James Dugundji: *Fixed point theory*. Springer Science & Business Media, 2013.

[91] Grazier, Kevin R., William I. Newman, James M. Hyman, Philip W. Sharp et David J. Goldstein: *Achieving Brouwer's law with high-order Stormer multistep methods*. ANZIAM Journal, 46:786–804, 2004.

[92] Guéneau, Armaël, Arthur Charguéraud et François Pottier: *A fistful of dollars: Formalizing asymptotic complexity claims via deductive program verification*. Dans *European Symposium on Programming*, pages 533–560. Springer, 2018.

[93] Guéneau, Armaël: *Procrastination, a proof engineering technique*. Coq Workshop 2018, avril 2018.

[94] Haberman, Richard: *Elementary applied partial differential equations*, tome 987. Prentice Hall Englewood Cliffs, NJ, 1983.

[95] Haftmann, Florian et Makarius Wenzel: *Constructive type classes in Isabelle*. Dans *International Workshop on Types for Proofs and Programs*, pages 160–174. Springer, 2006.

[96] Hairer, Ernst, Christian Lubich et Gerhard Wanner: *Geometric numerical integration illustrated by the Störmer–Verlet method*. Acta numerica, 12:399–450, 2003.

[97] Hairer, Ernst, Robert I McLachlan et Alain Razakarivony: *Achieving Brouwer's law with implicit Runge–Kutta methods*. BIT Numerical Mathematics, 48(2):231–243, 2008.

[98] Hairer, Ernst, Sylvert P. Norsett et Gerhard Wanner: *Solving Ordinary Differential Equations I: Nonstiff Problems*, tome 8. Springer-Verlag, 1993.

[99] Hairer, Ernst et Gerhard Wanner: *Examples of Stiff Equations*, pages 2–14. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. `https://doi.org/10.1007/978-3-642-05221-7_1`.

[100] Harrison, John: *Floating point verification in HOL light: The exponential function*. Dans Johnson, Michael (rédacteur): *Algebraic Methodology and Software Technology*, pages 246–260, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[101] Harrison, John: *A Machine-Checked Theory of Floating Point Arithmetic*. Dans Bertot, Yves, Gilles Dowek, André Hirschowitz, Christine Paulin et Laurent Théry (rédacteurs): *12th International Conference on Theorem Proving in Higher Order Logics, TPHOLs'99*, tome 1690 de *Lecture Notes in Computer Science*, pages 113–130, Nice, France, 1999. Springer-Verlag.

[102] Harrison, John: *Formal verification of floating point trigonometric functions*. Dans *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, pages 217–233, Austin, Texas, 2000.

[103] Harrison, John: *A HOL Theory of Euclidean space*. Dans Hurd, Joe et Tom Melham (rédacteurs): *18th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2005*, tome 3603 de *Lecture Notes in Computer Science*, Oxford, UK, 2005. Springer-Verlag.

[104] Harrison, John: *The HOL Light theory of Euclidean space*. Journal of Automated Reasoning, 50:173–190, 2013.

[105] Henrici, Peter: *Error propagation for difference methods*. Wiley, 1963.

[106] Higham, Nicholas J: *A survey of componentwise perturbation theory*, tome 48. American Mathematical Society, 1994.

[107] Higham, Nicholas J.: *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd édition, 2002.

[108] Higham, Nicholas J: *Functions of matrices: theory and computation*, tome 104. Siam, 2008.

[109] Hölzl, Johannes, Fabian Immler et Brian Huffman: *Type Classes and Filters for Mathematical Analysis in Isabelle/HOL*. Dans Blazy, Sandrine, Christine Paulin-Mohring et David Pichardie (rédacteurs): *Interactive Theorem Proving (ITP 2013)*, tome 7998 de *Lecture Notes in Computer Science*, pages 279–294. Springer, 2013.

[110] Howard, Paul et Jean E. Rubin: *Consequences of the Axiom of Choice*. Numéro v. 1. American Mathematical Society, 1998. `https://books.google.co.uk/books?id=ffXxBwAAQBAJ`.

[111] Huskey, Harry B.: *On the precision of a certain procedure of numerical integration*. Journal of Research of the National Bureau of Standards, 42, 1949.

[112] *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std 754-1985, 1985.

[113] *IEEE Standard for Floating-Point Arithmetic*. IEEE Std 754-2008, août 2008.

[114] Immler, Fabian: *Formally verified computation of enclosures of solutions of ordinary differential equations*. Dans *NASA Formal Methods Symposium*, pages 113–127. Springer, 2014.

[115] Immler, Fabian: *A Verified ODE Solver and the Lorenz Attractor*. Journal of Automated Reasoning, 61(1):73–111, juin 2018. `https://doi.org/10.1007/s10817-017-9448-y`.

[116] Immler, Fabian et Johannes Hölzl: *Numerical Analysis of Ordinary Differential Equations in Isabelle/HOL.* Dans Beringer, Lennart et Amy P. Felty (rédacteurs): *Third International Conference on Interactive Theorem Proving, ITP*, tome 7406 de *Lecture Notes in Computer Science*, pages 377–392. Springer, août 2012.

[117] Immler, Fabian et Christoph Traut: *The Flow of ODEs.* Dans Blanchette, Christian Jasmin et Stephan Merz (rédacteurs): *Proceedings of the 7th International Conference on Interactive Theorem Proving*, pages 184–199, Nancy, France, mars 2016. Springer International Publishing.

[118] Jeannerod, Claude Pierre et Siegfried M Rump: *Improved error bounds for inner products in floating-point arithmetic.* Society for Industrial and Applied Mathematics (SIAM) Journal on Matrix Analysis and Applications, 34(2):338–344, 2013.

[119] Jeannerod, Claude Pierre et Siegfried M. Rump: *On relative errors of floating-point operations: optimal bounds and applications.* Mathematics of Computation, 2016.

[120] Kalinina, Elizabeth A.: *The most precise computations using Euler's method in standard floating-point arithmetic applied to modelling of biological systems.* Computer Methods and Programs in Biomedicine, 111(2):471 – 479, 2013.

[121] Kaneko, Kunihiko: *Overview of coupled map lattices.* Chaos: An Interdisciplinary Journal of Nonlinear Science, 2(3):279–282, 1992.

[122] Kerjean, Marie et Assia Mahboubi: *A formal , classical proof of the Hahn-Banach theorem.* 2019.

[123] Knuth, Donald E.: *The Art of Computer Programming*, tome 2. Addison-Wesley, Reading, MA, USA, 3^{e} édition, 1998.

[124] Kornerup, Peter, Vincent Lefevre, Nicolas Louvet et Jean Michel Muller: *On the computation of correctly rounded sums.* IEEE Transactions on Computers, 61(3):289–298, mars 2012.

[125] Krebbers, Robbert et Bas Spitters: *Type classes for efficient exact real arithmetic in Coq.* Logical Methods in Computer Science, 9, juin 2011.

[126] Kulisch, Ulrich W.: *Advanced arithmetic for the digital computer - design of arithmetic units.* Springer, 2002.

[127] Kumar Das, Salu et Sandipan Roy: *Finite element analysis of aircraft wing using carbon fiber reinforced polymer and glass fiber reinforced polymer.* Journal Of Physics (IOP) Conference Series: Materials Science and Engineering, 402:012077, septembre 2018.

[128] Kutta, Wilhelm: *Beitrag zur naherungsweisen Integration totaler Differentialgleichungen.* Zeitschrift für angewandte Mathematik und Physik, 46:435–453, 1901.

[129] Lambert, John D.: *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem.* John Wiley & Sons, Inc., 1991.

[130] Lauter, Christoph Quirin: *Arrondi correct de fonctions mathématiques : fonctions univariées et bivariées, certification et automatisation*. Thèse de doctorat, 2008. `http://www.theses.fr/2008ENSL0482`, 2008ENSL0482.

[131] Le, Hung Q, William J Starke, J Stephen Fields, Francis P O'Connell, Dung Q Nguyen, Bruce J Ronchetti, Wolfram M Sauer, Eric M Schwarz et Michael T Vaden: *IBM power6 microarchitecture*. IBM Journal of Research and Development, 51(6):639–662, 2007.

[132] Lefèvre, Vincent, Damien Stehlé et Paul Zimmermann: *Worst Cases for the Exponential Function in the IEEE 754r decimal64 Format*. Dans Hertling, Peter, Christoph M. Hoffmann, Wolfram Luther et Nathalie Revol (rédacteurs): *Reliable Implementation of Real Number Algorithms: Theory and Practice*, pages 114–126, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[133] Lelay, Catherine: *How to express convergence for analysis in Coq*. Dans *The 7th Coq Workshop*, Sophia Antipolis, France, juin 2015.

[134] Lelay, Catherine: *Repenser la bibliothèque réelle de Coq : vers une formalisation de l'analyse classique mieux adaptée*. Thèse de Doctorat, Université Paris-Sud, juin 2015.

[135] Lelay, Catherine et Guillaume Melquiond: *Différentiabilité et intégrabilité en Coq. Application à la formule de d'Alembert*. Dans *23èmes Journées Francophones des Langages Applicatifs*, pages 119–133, Carnac, France, février 2012.

[136] Letouzey, Pierre: *A new extraction for Coq*. Dans *International Workshop on Types for Proofs and Programs*, pages 200–219. Springer, 2002.

[137] Leveson, Nancy G et Clark S Turner: *An investigation of the Therac-25 accidents*. Computer, 26(7):18–41, 1993.

[138] Lyapunov, Aleksandr Mikhailovich: *The general problem of the stability of motion*. International journal of control, 55(3):531–534, 1992.

[139] Magron, Victor, George A. Constantinides et Alastair F. Donaldson: *Certified Roundoff Error Bounds Using Semidefinite Programming*. CoRR, abs/1507.03331, 2015. `http://arxiv.org/abs/1507.03331`.

[140] Mahboubi, Assia: *The Rooster and the Butterflies*. Dans Carette, Jacques, David Aspinal, Christopher Lange, Petr Sojka et Wolfgang Windsteiger (rédacteurs): *CICM 2013 - Conference on Intelligent Computer Mathematics - 2013*, tome 7961 de *Lecture Notes in Artificial Intelligence*, pages 1–18, Bath, United Kingdom, juillet 2013. Springer. `https://hal.inria.fr/hal-00825074`.

[141] Mahboubi, Assia, Guillaume Melquiond et Thomas Sibut-Pinote: *Formally Verified Approximations of Definite Integrals*. Dans Blanchette, Jasmin Christian et Stephan Merz (rédacteurs): *Proceedings of the 7th Conference on Interactive Theorem Proving*, tome 9807 de *Lecture Notes in Computer Science*, pages 274–289, Nancy, France, août 2016.

[142] Mahboubi, Assia, Guillaume Melquiond et Thomas Sibut-Pinote: *Formally Verified Approximations of Definite Integrals*. Journal of Automated Reasoning, 62(2):281–300, 2019.

[143] Mahboubi, Assia et Enrico Tassi: *Canonical Structures for the Working Coq User*. Dans *4th International Conference on Interactive Theorem Proving, ITP*, pages 19–34, Rennes, France, juillet 2013.

[144] Mahboubi, Assia et Enrico Tassi: *Mathematical Components*, 2017. `https://math-comp.github.io/mcb/`.

[145] Mahmoud, Mohamed Yousri, Vincent Aravantinos et Sofiène Tahar: *Formalization of Infinite Dimension Linear Spaces with Application to Quantum Theory*. Dans *NASA Formal Methods NFM*, pages 413–427, Moffett Field, CA, USA, 2013. `https://doi.org/10.1007/978-3-642-38088-4_28`.

[146] Makarov, Evgeny et Bas Spitters: *The Picard Algorithm for Ordinary Differential Equations in Coq*. Dans Blazy, Sandrine, Christine Paulin et David Pichardie (rédacteurs): *ITP 2013, 4th Conference on Interactive Theorem Proving*, tome 7998 de *Lecture Notes in Computer Science*, pages 463–468, Rennes, France, juillet 2013. Springer.

[147] Martin-Dorel, Érik et Pierre Roux: *A reflexive tactic for polynomial positivity using numerical solvers and floating-point computations*. Dans *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 90–99. ACM, 2017.

[148] Mayero, Micaela: *Formalisation et automatisation de preuves en analyses réelle et numérique*. Thèse de doctorat, Université Paris VI, décembre 2001.

[149] Mayero, Micaela: *Using Theorem Proving for Numerical Analysis. Correctness Proof of al Automatic Differentiation Algorithm*. Dans *Proceedings of TPHOLs2002 (Theorem Proving in Higher Order Logics)*, tome 2410, page 246. Springer-Verlag Lecture Notes in Computer Science, 2002.

[150] Melquiond, Guillaume: *Proving bounds on real-valued functions with computations*. Dans Armando, Alessandro, Peter Baumgartner et Gilles Dowek (rédacteurs): *International Joint Conference on Automated Reasoning, IJCAR 2008*, tome 5195 de *Lecture Notes in Artifical Intelligence*, pages 2–17, Sydney, Australia, août 2008. Springer-Verlag. `https://hal.archives-ouvertes.fr/hal-00180138`.

[151] Moler, Cleve et Charles F. Van Loan: *Nineteen dubious ways to compute the exponential of a matrix*. Society for Industrial and Applied Mathematics (SIAM) review, 20(4):801–836, 1978.

[152] Moore, Eliakim H. et Herman L. Smith: *A General Theory of Limits*. American Journal of Mathematics, 44(2):102–121, 1922, ISSN 00029327, 10806377. `http://www.jstor.org/stable/2370388`.

[153] Moore, Gregory H: *The axiomatization of linear algebra: 1875-1940*. Historia Mathematica, 22(3):262–303, 1995.

[154] Moore, Ramon E: *Interval analysis*, tome 4. Prentice-Hall Englewood Cliffs, NJ, 1966.

[155] Muller, Jean Michel, Nicolas Brisebarre, Florent de Dinechin, Claude Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé et Serge Torres: *Handbook of Floating-Point Arithmetic*. Birkhäuser, 2010.

[156] Narkawicz, Anthony: *A Formal Proof Of The Riesz Representation Theorem*. Journal of Formalized Reasoning, 4(1):1–24, 2011. `https://doi.org/10.6092/issn.1972-5787/1952`.

[157] Naumowicz, Adam et Artur Korniłowicz: *A brief overview of Mizar*. Dans *International Conference on Theorem Proving in Higher Order Logics*, pages 67–72. Springer, 2009.

[158] Neher, Markus, Kenneth R Jackson et Nedialko S Nedialkov: *Validated solutions of initial value problems for ODEs*. Applied Mathematics and Computation, 105(1):21 – 68, 1999.

[159] Oberman, Stuart F: *Floating point division and square root algorithms and implementation in the AMD-K7/sup TM/microprocessor*. Dans *Proceedings 14th IEEE Symposium on Computer Arithmetic (Cat. No. 99CB36336)*, pages 106–115. IEEE, 1999.

[160] Owre, Sam, John M. Rushby et Natarajan Shankar: *PVS: A Prototype Verification System*. Dans *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, New York, USA*, pages 748–752, juin 1992. `https://doi.org/10.1007/3-540-55602-8_217`.

[161] Paşca, Ioana: *A Formal Verification for Kantorovitch's Theorem*. Dans *Journées Francophones des Langages Applicatifs*, janvier 2008. `http://hal.inria.fr/inria-00202808/en/`.

[162] Paşca, Ioana: *Formal Proofs for Theoretical Properties of Newton's Method*, 2010. `http://hal.archives-ouvertes.fr/inria-00463150/en/`, INRIA Research Report RR-7228.

[163] Paşca, Ioana: *Formal Verification for Numerical Methods*. Thèse de doctorat, novembre 2010. `http://tel.archives-ouvertes.fr/tel-00555158`.

[164] Paşca, Ioana: *Formally verified conditions for regularity of interval matrices*. Dans *International Conference on Intelligent Computer Mathematics*, pages 219–233. Springer, 2010.

[165] Paulson, Lawrence C: *The foundation of a generic theorem prover*. Journal of Automated Reasoning, 5(3):363–397, 1989.

[166] Pfenning, Frank et Christine Paulin-Mohring: *Inductively defined types in the Calculus of Constructions*. Dans *International Conference on Mathematical Foundations of Programming Semantics*, pages 209–228. Springer, 1989.

[167] Platzer, André: *The Complete Proof Theory of Hybrid Systems*. Dans *Logic in Computer Science*, pages 541–550. IEEE, 2012.

[168] Popiołek, Jan: *Introduction to Banach and Hilbert spaces—part III*. Journal of Formalized Mathematics, 3(199):1, 1991.

[169] Prévosto, Virgile et Franck Védrine: *Frama-Clang: a Frama-C front-end for C++*. EuroLLVM, avril 2014. `https://hal-cea.archives-ouvertes.fr/cea-01835663`, Poster.

[170] Quinlan, Gerald D.: *Round-off error in long-term orbital integrations using multistep methods*. Celestial Mechanics and Dynamical Astronomy, 58(4):339–351, 1994.

[171] Rademacher, Hans A: *On the accumulation of errors in processes of integration on high-speed calculating machines*. Dans *Proceedings of a Symposium on Large Scale Digital Calculating Machinery, Annals of the Computer Laboratory of Harvard University*, tome 16, pages 176–185, 1948.

[172] Ramos, Juan I.: *Linearization methods in classical and quantum mechanics*. Computer Physics Communications, 153(2):199–208, 2003.

[173] Revol, Nathalie, Kyoko Makino et Martin Berz: *Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY*. The Journal of Logic and Algebraic Programming, 64(1):135 – 154, 2005.

[174] Richardson, Lewis Fry: *The approximate arithmetical solution by finite differences with an application to stresses in masonry dams*. Philosophical Transactions of the Royal Society of America, 210:307–357, 1911.

[175] Richtmyer, Robert D. et Keith W. Morton: *Difference methods for initial-value problems*. Interscience Publishers, 1967.

[176] Rouhling, Damien: *A Formal Proof in Coq of a Control Function for the Inverted Pendulum*. Dans *CPP 2018 - 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 1–14, Los Angeles, United States, janvier 2018. `https://hal.inria.fr/hal-01639819`.

[177] Roux, Pierre: *Formal Proofs of Rounding Error Bounds - With Application to an Automatic Positive Definiteness Check*. Journal of Automated Reasoning, 57(2):135–156, 2016. `https://doi.org/10.1007/s10817-015-9339-z`.

[178] Rudin, Walter: *Real and complex analysis*. Tata McGraw-hill education, 2006.

[179] Rump, Siegfried M, Florian Bünger et Claude Pierre Jeannerod: *Improved error bounds for floating-point products and Horner's scheme*. BIT Numerical Mathematics, 56(1):293–307, 2016.

[180] Rump, Siegfried M et Claude Pierre Jeannerod: *Improved backward error bounds for LU and Cholesky factorizations*. Society Journal on Matrix Analysis and Applications, 35(2):684–698, 2014.

[181] Runge, Carl: *Über die numerische Auflösung von Differentialgleichungen*. Mathematische Annalen, 46(2):167–178, 1895.

[182] Russinoff, David M: *A Mechanically Checked Proof of IEEE Compliance of the Floating Point Multiplication, Division and Square Root Algorithms of the AMD-K7™ Processor*. London Mathematical Society (LMS) Journal of Computation and Mathematics, 1:148–200, 1998.

[183] Sandretto, Julien Alexandre dit et Alexandre Chapoutot: *Validated Explicit and Implicit Runge-Kutta Methods*. Reliable Computing, 22, 2016.

[184] Sawada, Joe: *Formal verification of divide and square root algorithms using series calculation*. Dans *ACL 2002*, 2002.

[185] Scholz, Gudrun et Fritz Scholz: *First-order differential equations in chemistry*. ChemTexts, 1(1):1, novembre 2014. `https://doi.org/10.1007/s40828-014-0001-x`.

[186] Shidama, Yasunari: *Banach space of bounded linear operators*. Formalized Mathematics, 12(1):39–48, 2004.

[187] Sibut Pinote, Thomas: *Investigations in Computer-Aided Mathematics: Experimentation, Computation, and Certification*. Thèse de doctorat, 2017.

[188] Skeel, Robert D.: *Symplectic integration with floating-point arithmetic and other approximations*. Applied Numerical Mathematics, 29(1):3 – 18, 1999.

[189] Smith, Gordon D.: *Numerical solution of partial differential equations: finite difference methods*. Oxford University Press, 1985.

[190] Smojver, Ivica et Darko Ivančević: *Application of numerical methods in the improvement of safety of aeronautical structures*. Transportation Research Procedia, 28:164–172, 2017.

[191] Sozeau, Matthieu et Nicolas Oury: *First-class type classes*. Dans *International Conference on Theorem Proving in Higher Order Logics*, pages 278–293. Springer, 2008.

[192] Spijker, Marc N.: *Error propagation in Runge-Kutta methods*. Applied Numerical Mathematics, 22:309–325, 1996.

[193] Spitters, Bas: *Constructive and intuitionistic integration theory and functional analysis*. Thèse de doctorat, University of Nijmegen, 2002.

[194] Spivak, Michael D.: *A Comprehensive Introduction to Differential Geometry*. Numéro vol. 5 dans *A Comprehensive Introduction to Differential Geometry*. Publish or Perish, 1975. `https://books.google.fr/books?id=W4JiuwEACAAJ`.

[195] Sterbenz, Pat H.: *Floating point computation*. Prentice Hall, 1974.

[196] Sterne, Theodore E.: *The Accuracy of Numerical Solutions of Ordinary Differential Equations*. Mathematical Tables and Other Aids to Computation, 7(43):159–164, 1953.

[197] Tatum, Jeremy B.: *Physics topics: Electricity and Magnetism, 2007, chapter 3*. Free online textbook.

[198] Taylor, Charles A., Thomas JR Hughes et Christopher K. Zarins: *Finite element modeling of blood flow in arteries*. Computer methods in applied mechanics and engineering, 158(1-2):155–196, 1998.

[199] Tendler, Joel M, J Steve Dodson, JS Fields, Hung Le et Balaram Sinharoy: *POWER4 system microarchitecture*. IBM Journal of Research and Development, 46(1):5–25, 2002.

[200] Trybulec, Andrzej: *Non-Negative Real Numbers. Part I1*. Journal of Formalized Mathematics, 1998.

[201] Tucker, Warwick: *A rigorous ODE solver and Smale's 14th problem*. Foundations of Computational Mathematics, 2(1):53–117, 2002.

[202] US Government Accountability Office: *Defense Patriot Missile: Software problem led to system failure at Dhahran, Saudi Arabia*. US Government Accountability Office Reports, rapport no. GAO/IMTEC-92-26, 1992.

[203] Wanner, Gerhard et Ernst Hairer: *Solving ordinary differential equations II*. Springer Berlin Heidelberg, 1996.

[204] Weil, André: *Sur les espaces à structure uniforme et sur la topologie générale*. Publications de l'Institut de mathématique de l'Université de Strasbourg. Hermann & cie, 1938. `https://books.google.fr/books?id=qVfvAAAAMAAJ`.

[205] Wenzel, Makarius: *Isabelle/Isar—a generic framework for human-readable proof documents*. From Insight to Proof—Festschrift in Honour of Andrzej Trybulec, 10(23):277–298, 2007.

[206] Zienkiewicz, Olek C., Robert L. Taylor et Jin Zhou Zhu: *The finite element method: its basis and fundamentals*. Elsevier/Butterworth Heinemann, Amsterdam, seventh édition, 2013. `http://dx.doi.org/10.1016/B978-1-85617-633-0.00001-0`.

Annexe A

Correspondance entre principaux énoncés papier et Coq

Énoncé papier	Fichier Coq	Énoncé Coq
Théorème 4.3	Average10/Average2n.v	average10_correct
Lemme 6.1	CoqRK/Norms.v	mx_vec_norm_submult
Lemme 6.2	CoqRK/Norms.v	mx_norm_submult
Lemme 6.3	CoqRK/FP_prel.v	dot_product_error
Théorème 6.4	CoqRK/FP_prel.v	mx_vec_prod_error
Théorème 6.5	CoqRK/FP_prel.v	mx_prod_error
Lemme 6.6	CoqRK/FP_prel.v	error_mat_const_prod
Lemme 6.7	CoqRK/FP_prel.v	bounds_alpha_bound_plus_flt
Lemme 6.8	CoqRK/FP_prel.v	build_bound_mult_loc
Corollaire 6.9	CoqRK/FP_prel.v	build_bound_mult_loc_wk
Lemme 6.10	CoqRK/Instanciations.v	Euler_loc_FLT
Lemme 6.11	CoqRK/Instanciations.v	RK2_loc_FLT
Théorème 6.12	CoqRK/Error_loc_to_glob.v	error_loc_to_glob
Théorème 6.13	CoqRK/Instanciations.v	Euler_glob_FLT
Théorème 6.14	CoqRK/Instanciations.v	RK2_glob_FLT
Lemme 7.1	CoqLM/hilbert.v	parallelogram_id
Lemme 7.2	CoqLM/hilbert.v	Cauchy_Schwartz
Théorème 7.3	CoqLM/hilbert.v	ortho_projection_subspace
Théorème 8.1	CoqLM/fixed_point.v	FixedPoint_C_phi
Théorème 8.2	CoqLM/lax_milgram.v	Riesz_Frechet_strong_phi
Théorème 8.4	CoqLM/lax_milgram.v	Lax_Milgram_phi
Lemme 8.5	CoqLM/lax_milgram_cea.v	Lax_Milgram_Cea
Lemme 8.6	CoqLM/finite_dim.v	EVN_comp_m
Lemme 8.7	CoqLM/finite_dim.v	span_closed
Lemme 8.8	CoqLM/finite_dim.v	sum_span_orth_compl_complete
Théorème 8.9	CoqLM/finite_dim.v	EVN_closed

Annexe B

Scripts Gappa pour borner l'erreur des coefficients en $h\lambda$

Fichier `euler.gappa`

```
@rnd = float<ieee_64,ne>;

lt = rnd(l);
r rnd= h*lt;
re = h*l;

{ h*l in [-2,-1b-100] /\ h in [1b-60,1]
   -> r -/ re in [-2049b-63,2049b-63] /\
      r - re in [-4b-53, 4b-53]
}
```

Fichier `rk2.gappa`

```
@rnd = float<ieee_64,ne>;

lt = rnd(l);
w rnd= h*h*0.5*lt*lt;
we = h*h*0.5*l*l;

{ h*l in [-2,-1b-100] /\ h in [1b-60,1]
   ->  w -/ we in [-10241b-64,10241b-64] /\
       w - we in [-20481b-64,20481b-64]
}
we -> ((h*l)*(h*l))*0.5;
```

Fichier `rk4.gappa`

```
@rnd = float<ieee_64,ne>;

lt = rnd(l);

w1 rnd= h/6*lt;
we1 = h/6*l;

w2 rnd= h/3*lt;
we2 = h/3*l;

w3 rnd= h*h/6*lt*lt;
we3 = h*h/6*l*l;

w4 rnd= h*h*h/12*lt*lt*lt;
we4 = h*h*h/12*l*l*l;

w5 rnd= h*h*h*h/24*lt*lt*lt*lt;
we5 = h*h*h*h/24*l*l*l*l;


{ h*l in [-2,-1b-100]
 /\ h in [1b-60,1]
   -> w1 -/ we1 in [-6145b-64,6145b-64] /\
      (w1-we1) in [-1b-53,1b-53] /\
      w2 -/ we2 in [-6145b-64,6145b-64] /\
      (w2-we2) in  [-2b-53,2b-53] /\
      w3 -/ we3 in  [-12289b-64,12289b-64]  /\
      (w3-we3) in  [-4b-53,4b-53]  /\
      w4 -/ we4 in  [-18433b-64,18433b-64] /\
      (w4-we4) in  [-6b-53,6b-53] /\
      w5 -/ we5 in [-24577b-64,24577b-64] /\
      (w5-we5) in  [-8b-53,8b-53]
}

we1 -> (h*l)/6;
we2 -> (h*l)/3;
we3 -> (h*l)*(h*l)/6;
we4 -> (h*l)*(h*l)*(h*l)/12;
we5 -> (h*l)*(h*l)*(h*l)*(h*l)/24;
```

Table des figures

Liste des tableaux

Index

C

D

E

F

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Formalisations d'analyses d'erreurs en analyse numérique et en arithmétique à virgule flottante

Mots clés : Arithmétique à virgule flottante, Erreurs d'arrondi, Preuves formelles, Coq, Runge-Kutta

Résumé : Cette thèse est constituée de trois contributions liées à la formalisation en Coq d'analyses d'erreurs dans les domaines de l'analyse numérique et de l'arithmétique à virgule flottante.
Nous avons tout d'abord proposé un algorithme calculant la moyenne de deux nombres flottants décimaux et avons montré que cet algorithme fournissait l'arrondi correct. Nous avons formalisé l'algorithme et sa preuve de correction dans l'assistant de preuves Coq.
La seconde contribution de la thèse est l'analyse et la formalisation de bornes sur les erreurs d'arrondi d'implémentations de méthodes de Runge-Kutta appliquées à des systèmes linéaires. Nous avons proposé une méthodologie générique permettant de construire une borne sur l'erreur d'arrondi accumulée au cours des itérations et qui tient compte d'éventuels dépassements de capacité. Nous avons ensuite appliquée la méthodologie à des méthodes de Runge-Kutta classiques, comme les méthodes d'Euler et de RK2. Nous avons proposé une formalisation de l'analyse, incluant la définition de normes matricielles, la démonstration de bornes sur les erreurs d'arrondi d'opérations matricielles et la formalisation des résultats génériques et de leur application.
Enfin, nous avons proposé la formalisation de résultats d'analyse fonctionnelle qui servent de fondements à la méthode des éléments finis. Cette formalisation repose sur la bibliothèque Coquelicot et inclut la théorie des espaces de Hilbert, la formalisation du théorème de Lax–Milgram et la preuve de complétude des sous-espaces de dimension finie d'espaces de Hilbert.

Title : Formalizations of error analysis in numerical analysis and floating-point arithmetic

Keywords : Floating-point arithmetic, Rounding errors, Formal proofs, Coq, Runge-Kutta

Abstract : This thesis consists of three contributions related to the Coq formalization of error analysis in numerical analysis and floating-point arithmetic.
First, we have exhibited an algorithm computing the average of two decimal floating-point numbers and have proved that this algorithm computes the correct rounding. We have formalized the algorithm and its correctness proof in the Coq proof assistant.
The second contribution of the thesis is the analysis and the formalization of rounding error bounds associated to the implementation of Runge-Kutta methods applied to linear systems. We have proposed a generic methodology to build a bound on the error accumulated over the iterations, taking potential underflow and overflow into account. We have then instantiated this methodology for classic Runge-Kutta methods, *e.g.* the Euler and RK2 methods. We have proposed a formalization of the results, including the definition of matrix norms, the proof of rounding error bounds for matrix operations and the formalization of the generic results and their instantiations.
Finally, we have proposed a formalization of functional analysis results that serve as foundations for the finite element method. This formalization is based on the Coquelicot library and includes the theory of Hilbert spaces, the formal proof of the Lax–Milgram Theorem and the proof of completeness of finite dimensional subspaces of Hilbert spaces.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France