Amélioration de la rapidité d'exécution des systèmes EDO de grande taille issus de Modelica

Thibaut-Hugues Gallois

NNT : 2015SACLC023

THESE DE DOCTORAT
DE L'UNIVERSITE PARIS-SACLAY,
préparée à CentraleSupelec

ÉCOLE DOCTORALE N° 573
INTERFACES

Mathématiques Appliquées

Par

M. Thibaut-Hugues Gallois

Amélioration de la rapidité d'exécution des systèmes EDO de grande taille issus de Modelica

Thèse présentée et soutenue à Toulon, le 3 décembre 2015

Composition du Jury :

M. Agélas Léo	Ingénieur de recherche, IFPEN	Encadrant
M. Angot Philippe	Professeur, Université de Aix-Marseille	Rapporteur
M. Ben Gaïd Mongi	Ingénieur de recherche, IFPEN	Examinateur
M. Brac Jean	Directeur de recherche, IFPEN	Encadrant
M. Galusinski Cédric	Professeur, Université de Toulon	Président
M. Hammami Omar	Professeur, ENSTA ParisTech	Examinateur
M. Pironneau Olivier	Professeur, Université Pierre et Marie Curie	Rapporteur
M. Soriano Thierry	Professeur, Université de Toulon, SEATECH	Directeur de thèse

Ecole Centrale Paris
Grande Voie des Vignes
92295-Châtenay-Malabry

IFPEN
1-4 Avenue de Bois-Préau
92852 Rueil-Malmaison

A mes parents,

Remerciements

La page des remerciements, bien que souvent rédigée en dernier, est certainement l'une des pages les plus parcourues et je vais donc m'essayer à ce travail de rédaction en conjuguant exhaustivité et justesse.

Plus jeune, essayant de situer une date par rapport à une autre lors d'une épreuve sur table d'Histoire, je commençais toujours par le commencement, situant les dates les unes après les autres et je ne dérogerai pas à cette règle ici. Avant cette thèse, il y eut un stage que m'avait fait découvert Antoine Henrot, Professeur à l'Ecole des Mines de Nancy. Ce stage, réalisé à IFP Energies Nouvelles, a été encadré par Jean Brac dont l'enthousiasme m'a poussé à accepter à bras ouverts la thèse qu'il me tendait. Jean est une personne débordante d'idées, toujours prête à explorer d'autres pistes. J'ai découvert en moi ce goût pour la recherche grâce à Jean et pour cela, je le remercie très chaleureusement. Par la suite, j'ai fait la connaissance de mon directeur de thèse, Thierry Soriano, que j'ai appris à connaître et à apprécier aux cours de séjours à Toulon et surtout de séjours en Suède, qui resteront, je pense, gravés dans nos mémoires à tous. Thierry aime la science et est également d'une gentillesse exceptionnelle et pour tout cela, je le remercie tout particulièrement. En troisième année, après un départ en retraite bien mérité, Jean a laissé sa place à Léo Agélas. J'ai passé une merveilleuse troisième année en compagnie de Léo (nous avons en particulier découvert les sandwichs à trous, autrement appelés Bagels) et je tenais également à le remercier très chaleureusement pour toute la richesse scientifique et humaine qu'il m'a apporté. Ses encouragements constants et son dévouement m'ont beaucoup aidé. Je tiens également à remercier Mongi pour ces précieux conseils, sa disponibilité et son soutien pendant ces trois années. Je remercie également Philippe Angot et Olivier Pironneau pour avoir accepté d'être rapporteurs de ma thèse et ainsi d'avoir pris le temps de lire mon manuscrit. Enfin, merci à Cédric Galusinski et Omar Hammami d'avoir accepté de participer à mon jury de soutenance.

Ma thèse s'étant déroulé à 99.9 % du temps dans les bureux d'IFP Energies Nouvelles, il est tout naturel que je remercie un certain nombre de personnes ici. Tout d'abord, je remercie tout particulièrement Zakia Benjelloun-Touimi, chef du département de mathématiques appliquées, sans qui rien n'aurait été possible. Je la remercie également pour tout ses petits mots d'encouragements et sa gentillesse. Je remercie également les différentes secrétaires, Virginie, Nelly et Sylvie, qui ont toujours su organiser parfaitement mes différents déplacements ainsi que la vie quotidienne à IFP Energies Nouvelles. Merci à tous mes collègues mathématiciens et informaticiens, en particulier Huy, Isabelle, Stéphane, Julien, Sylvie P., Miguel, Delphine, Frédéric, Françoise, Anthony, Aziz, Benjamin, Julien, Thomas, Ani, Olivier, Henry, Guillaume, Long, Jean-Yves, Jean-Louis, Steven, Christophe D. et Sylvie W.. J'en viens maintenant à mes compagnons de thèse, avec qui j'ai partagé tant de déjeuners et parfois de dîners. Je pense tout d'abord à ceux que j'ai connu en arrivant et qui m'ont si bien accueilli, Antoine, Eugénio, Claire (qui va bientôt soutenir sa thèse, tous mes encouragements du fond de mon coeur vont vers elle), Tassadit, Brahim, Soleiman,

Carole, Ratiba, Simon (avec qui j'ai partagé tant d'intéressantes discussions scientifiques) et Franck (qui m'a supporté quelques mois dans le bureau). Ensuite viennent les doctorants de ma génération, Pierre et Benoît avec lesquels les formations n'auraient pas eu la même saveur. Puis vient la jeune génération, Aboubacar, Huong (merci pour tous ces merveilleux gâteaux d'anniversaire), Riad (le thésard le plus connaisseur de foot que je connaisse), Mohamed, Ivana et Nicolas (merci pour ces expéditions au bout du monde). Merci à vous tous pour tous ces moments de joie qui m'ont aidé à tenir dans les moments difficiles, nos déjeuners le midi vont énormément me manquer.

J'en viens à mes amis, et plus particulièrement Pierre-Emmanuel, Stéphane, Maxime, Julien, Mathieu, Sylvain, Rémi, Thomas, Philippe, Olivier, Alexandre, merci d'avoir été là, à mes cotés. Ce n'est pas toujours facile de se voir, mais les moments passés ensemble sont précieux. Je remercie également ma douce amie qui m'a tant aidé à aller jusqu'au bout de cette thèse par ses encouragements et sa présence.

Enfin, je remercie ma famille pour leurs encouragements et plus particulièrement mes tantes Marie-Christine et Brigitte, mon oncle Ronald, mes cousines Anne-Sophie et Marie et mon cousin Harold. Je pense également à ma très chère grand-mère, qui aurait tant aimé voulu voir l'accomplissement de mes études, et à mon merveilleux et si fort grand-père. Enfin du fond de mon coeur, merci maman et merci papa, votre éducation et vos valeurs m'ont énormément apporté et si je peux écrire cette dernière phrase c'est grâce à votre soutien, vos sacrifices et votre amour.

Résumé

Résumé

L'étude des systèmes aux équations différentielles ordinaires vise à prédire le futur des systèmes considérés. La connaissance de l'évolution dans le temps de toutes les variables d'état du modèle permet de prédire de possibles changements radicaux des variables ou des défaillances, par exemple, un moteur peut exploser, un pont peut s'écrouler, une voiture peut se mettre à consommer plus d'essence. La résolution des équations différentielles ordinaires est alors une étape essentielle dans la construction des systèmes physiques en terme de dimensionnement et de faisabilité. Le solveur de tels systèmes EDOs doit être rapide, précis et pertinent.

Ces modèles peuvent être construits en utilisant un langage de modélisation tel que Modelica, qui décrit l'évolution d'un système physique au moyen des équations différentielles ordinaires. Les coefficients de l'EDO peuvent être introduits comme des variables plus ou moins complexes ; leurs valeurs sont gelées à chaque simulation. Lors de l'étude du dimensionnement du système, le nombre de simulations est généralement grand étant donné que les simulations sont toujours moins coûteuses que les expérimentations. De plus, dans de nombreux cas, afin de contrôler le système le plus longtemps possible, on a besoin d'un temps de simulation très grand. Un nombre important de simulation est également requis afin de balayer l'espace des paramètres de manière pertinente. C'est pourquoi, la simulation mathématique doit être très efficace. Par ailleurs, l'étude du comportement du système dans l'espace des paramètres peut être piloté par des algorithmes qui ne sont pas développés dans cette thèse.

En pratique, il n'est pas possible de trouver une fonction continue qui soit solution exacte du problème EDO. C'est pourquoi, des méthodes numériques sont utilisées afin de donner des solutions discrètes qui approchent la solution continue avec une erreur contrôlable. La gestion précise de ce contrôle est très important afin d'obtenir une solution pertinente en un temps raisonnable.

Notre but est de réduire le coût de simulation afin de permettre le plus grand nombre de simulations en un temps raisonnable. En effet, les systèmes dynamiques peuvent contenir des dérivées spatiales et leur discrétisation peut ajouter un très grand nombre d'équations. Il devient alors nécessaire de résoudre de tels systèmes de manière efficace et de fournir une solution précise en quelques jours au lieu de quelques mois.

Cette thèse développe un nouveau solveur qui utilise plusieurs méthodes d'amélioration de la vitesse d'exécution des systèmes EDOs. La première méthode est l'intégration du système EDO avec un schéma numérique qui soit stable et robuste. Un nouveau schéma est proposé dans cette thèse. Le but est de minimiser le coût de l'intégration en produisant une erreur qui soit le plus proche possible de la tolérance maximale permise par l'utilisateur du solveur. Une autre méthode pour améliorer la vitesse d'exécution est de paralléliser le solveur EDO en utilisant une architecture multicoeur et multiprocesseur.

Enfin, le solveur a été testé avec différentes applications d'OpenModelica afin de comparer son efficacité avec les solveurs classiques tels que DASSL. Un exemple à évènements et plusieurs systèmes de grande taille ont été en particulier testés.

Mots-clefs

Improvement of the execution speed of large scale ODEs systems from Modelica

Abstract

The study of systems of Ordinary Differential Equations aims at predicting the future of the considered systems. The access to the evolution of all states of a system's model allows us to predict possible drastic shifts of the states or failures, e.g. an engine blowing up, a bridge collapsin, a car consuming more gasoline etc. Solving ordinary differential equations is then an essential step of building industrial physical systems in regard to dimensioning and reliability. The solver of such ODE systems needs to be fast, accurate and relevant.

These models can be built using a modelling language such as Modelica, which describes the evolution of a physical system by means of ordinary differential equations. The coefficients of such ODE can be introduced as more or less complexe variables; their values are frozen at each simulation. In the study of system dimensioning, the number of simulations is usually large since simulations are always less expensive than experimentation. Moreover, in many cases, to control the system within a large time horizon needs a long time of simulations; a large number of simulations is also required in order to sweep the parameter parameter space in a relevant way. Therefore, the simulation mathematics requires to be very efficient. On a side note, the study of the system behaviour through the parameter space can be driven by algorithms which are note in the scope of my thesis.

In practice, it is not possible to find a continuous function as the exact solution of the real ODE problem. Consequently numerical methods are used to give discrete solutions which approximates the continuous one with a controllable error. The correct handline of this control is very important to get a relevant solution within an acceptable recovery time. Starting from existing studies of local and global errors, this thesis work goes more deeply and adjusts the time step of the integration time algorithm and solves the problem in a very efficient manner.

Our goal is to reduce the simulation cost in order to make a highest possible number of simulations during a reasonable time. Indeed, dynamic systems might contain spatial derivatives and they can be hidden into the considered ordinary differential systems by adding a large number of discretized equations. It becomes necessary to efficiently solve such large systems and to provide an accurate solution within days instead of months.

This thesis offers a new solver which uses several methods to improve the execution speed of ODE systems. One method is the integration of ODE systems with a numerical

scheme both fast and robust. A new scheme is proposed is this thesis, to minimize the cost of integration. Another method to improve the execution speed is to parallelize the ODE solver by using a multicore and a multiprocessor architecture.

Finally, the solver has been tested with different applications from OpenModelica in order to compare their efficiency with well-known solvers as DASSL. An example with several events and a large scale system has been tested among others.

Models built from a modelling language such as Modelica which describes the evolution of a physical system in time contain ordinary differential equations that need integrating in order to show the behaviour of the state variables of the system. Practically, it is not often possible to find a continuous function which is the exact solution of ODE problems. Therefore numerical methods are used to give discrete solutions which approximate the continuous one with a controllable error. The more state variables there are to integrate, the more important the time of integration. In the study of parameters variation, the number of simulations is important, so we need to reduce the cost in order to have the highest number of simulations during a reasonable time.

Two methods have been developped in this thesis to improve the execution speed of ODE systems.

The first method is the integration of the ODE system with a numerical scheme which is fast and robust at the same time. The goal is to minimize the cost of integration by produce an error lower than the tolerance given by the user.

The second method to improve the execution speed is to parallelize the ODE solver by using a multicore and a multiprocessor architecture.

Finally, the methods have been tested with different applications in order to compare their efficiency with traditionnal methods. An example with several events and a large scale system has been tested among others.

Keywords

Table des matières

Introduction

La chimie, la biologie, l'ingénierie, la physique sont autant de domaines parmi d'autres traitant de problèmes qui contiennent des systèmes aux équations différentielles ordinaires de la forme (1).

$$\begin{cases} t \in [0, T], \\ \dfrac{\mathrm{d}\bar{y}}{\mathrm{d}t} = f(\bar{y}), \\ \bar{y}(0) = y_0, \end{cases} \tag{1}$$

avec $\bar{y} \in \mathbb{R}^d$, $f \in C^p(\mathbb{R}^d, \mathbb{R}^d)$.

La résolution de ces problèmes permet de connaître l'évolution dans le temps des variables du système et ainsi de prédire le comportement du modèle décrit par les équations du système. Prédire le comportement d'un modèle est très important car cela permet de détecter un éventuel dysfonctionnement, une rupture mécanique ou un arrêt d'un moteur par exemple.

En pratique, il n'est souvent pas possible de déterminer une fonction continue qui soit solution exacte du problème EDO. On a alors recours à des méthodes numériques d'intégration permettant de donner une solution discrète qui approche la solution continue avec une erreur.

Le système EDO peut être complexe, non linéaire, couplé, hybride en temps, plus ou moins raide, plus ou moins difficile à initialiser. De plus, certains de ces systèmes physiques, comme la dynamique des fluides en milieu poreux, en atmosphère ou dans les océans, l'évolution de certains matériaux sont issus de discrétisations spatiales des dérivées partielles (EDPs). Dans les cas où la dicrétisation est fine, la taille de l'espace est grande et les systèmes d'EDOs peuvent alors être de grande taille ($10^4, 10^6$ équations) et donner des temps de calcul très longs. En effet, plus il y a de variables d'état à intégrer, plus le coût de l'intégration numérique en termes de temps de calcul est important. Il est primordial d'effectuer la résolution de tels systèmes EDO de la manière la plus rapide qu'il soit. En effet, dans le cas où l'on cherche à détecter un éventuel dysfonctionnement dans un modèle et à ainsi prédire le futur du système, on peut avoir besoin de simuler les modèles pendant un temps très grand. Ainsi une méthode de résolution rapide permettra de réduire le temps d'exécution et de permettre plus de simulations dans un temps plus court. Cette répétition des simulations est très utile dans le cadre de la variation de paramètres, où on a besoin de simuler plusieurs fois le même modèle en changeant la valeur d'un paramètre. Cette thèse va s'attacher à accélérer la résolution des systèmes EDO de grande taille tout en garantissant la plus grande stabilité face aux systèmes raides.

Dans le cas des modèles non linéaires, la raideur du système (1) peut devenir grande si une ou plusieurs valeurs propres de la matrice Jacobienne $J = \frac{\partial f}{\partial \bar{y}}$ a une partie réelle négative grande. La présence de dynamiques rapides et lentes pose de nombreuses diffi-

cultés aux solveurs numériques utilisant des schémas explicites. En effet, la condition de stabilité de Courant-Friedrichs-Lewy (condition CFL) impose un pas de temps maximal qui correspond à l'échelle de temps la plus petite du système. L'ensemble de ces raisons conduit les numériciens à utiliser des schémas implicites en temps pour simuler les EDPs de grande taille évoluant dans le temps. Les schémas implicites les plus communéments utilisés sont les schémas de type Backward Differentiation Formulas (BDF) (voir [BHJB77, IHAR09, SD80]). Le schéma BDF est une méthode multipas implicite qui utilise les points connus calculés à différents temps précédents ainsi que la solution inconnue du point courant. L'ordre du terme d'erreur peut être facilement ajusté en utilisant plus ou moins de points précédents. Cette méthode est connue pour fournir une bonne stabilité de solution pour la résolution de systèmes raides pour laquelle une méthode implicite est indispensable. Une résolution implicite en temps nécessite la résolution à chaque pas de temps d'un système d'équations non linéaire de grande taille qui couple toutes les variables. De plus, la résolution d'un système non linéaire requiert le calcul des matrices Jacobiennes pour chaque pas de temps. Pour la plupart des systèmes, la Jacobienne exacte peut être coûteuse à calculer et difficile à obtenir, à cause par exemple de la taille du modèle et l'utilisation de schémas de discrétisation complexes.

La première partie de cette thèse qui se trouve dans le Chapitre 2 consiste à améliorer le schéma BDF en proposant une nouvelle classe de schémas que nous appellerons Linearized Interpolation Backward Differentiation Formulas (LIBDF). Ces méthodes sont linéairement implicites ce qui permet de n'avoir besoin de résoudre qu'un système linéaire par itération de temps, au contraire du schéma BDF pour lequel il faut résoudre un système non linéaire. De plus, dans le cas linéaire (ce qui signifie que f est linéaire), le schéma LIBDF dégénère en schéma BDF et ainsi tous les critères d'un bon schéma numérique (consistance, ordre, zero-stabilité, stabilité absolue et convergence) sont vérifiés.

Un intérêt tout particulier dans ce travail est l'utilisation des points d'équilibre du système (1). En effet ,nous montrons dans un nouveau théorème que dans le cas d'équilibres stables, l'erreur globale ne dépend pas du temps même dans le cas non linéaire. De plus, ce majorant n'explose pas quand la raideur du problème devient très grande ce qui en fait un majorant de l'erreur globale très utile.

Nous montrons également, toujours dans la première partie de cette thèse, une nouvelle méthode de gestion des discontinuités lors de la résolution des systèmes EDOs hybrides, c'est-à-dire des systèmes EDOs qui mèlent temps continu et discret. Ces systèmes sont particulièrement complexes à simuler et peuvent être coûteux dans le cas où le nombre de discontinuités est élevé, il est ainsi important que la nouvelle méthode soit rapide et économique en terme de coût de calcul.

Le calcul du premier point de la solution est également très important et la première partie de la thèse montre deux nouvelles méthodes efficaces, stables et rapides afin de calculer ce premier point en n'utilisant que la condition initiale du problème.

Enfin on développe une méthode de calcul appelée Waveform Relaxation, qui s'appuie sur un découpage du modèle et sur l'intégration séparée des différentes parties du modèle.

Les récentes années ont vu le développement de méthodes parallèles efficaces pour la résolution numérique des systèmes EDOs. En effet, le besoin de solveurs plus rapides a conduit à utiliser des ordinateurs en parallèle et des systèmes distribués. Le principal défi est de tirer bénéfice de cet énorme potentiel de puissance de calcul. Afin d'être efficace, un tel solveur doit s'appuyer sur des algorithmes qui sont bien adaptés à ces nouvelles architectures. Ainsi la seconde partie de cette thèse (le Chapitre 3) s'attache à paralléliser le schéma numérique LIBDF développé dans la première partie. On montre en particulier

quelles sont les parties les plus coûteuses en termes de temps de calcul et sur lesquelles il est important de concentrer la parallélisation. Deux parallélisations sont ainsi proposées :

- une parallélisation qui utilise une architecture de calcul en mémoire partagée en utilisant des directives OpenMP,
- une parallélisation qui utilise une architecture de calcul en mémoire distribuée en utilisant des fonctions de la blibothèque MPI.

Le langage Modelica s'est largement imposé pour résoudre ce genre de problème EDO. C'est en effet un langage acausal, orienté objet, créé pour formuler la phénoménologie de sous-systèmes physiques entre eux. Ainsi, la troisième partie de la thèse, le Chapitre 4, décrit différents modèles construits en langage Modelica et pour lesquelles on étudie les nouveaux schémas numériques ainsi que les techniques de parallélisation décrits dans les deux premières parties. Les modèles sont : la diffusion de la chaleur dans une tige, l'écoulement de Saint-Venant, la balle rebondissante (avec et sans déformation) et l'écoulement diphasique huile-eau en tenant compte de la pression capillaire.

Abréviations

- BDF : Backward Differentiation Formula
- CATIA : Conception Assistée Tridimensionnelle Interactive Appliquée
- CSR : Compressed Sparse Row
- CSC : Compressed Sparse Column
- condition CFL : condition de Courant Friedrichs Lewy
- DASSL : Differential Algebraic System SoLver
- DIRK : Diagonally Implicit Runge Kutta
- EDO(s) : Equation(s) Différentielle(s) Ordinaire(s)
- EDP(s) : Equation(s) aux Dérivées Partielles
- FLC : Statégie du coefficient fixe unique
- FMI : Functionnal Mock-up Interface
- FMU : Functionnal Mockup Unit
- HOT : Higher Order Terms
- INT : Stratégie à coefficients fixes
- IRK : Implicit Runge Kutta
- IVP : Initial Value Problem
- LIBDF : Linearized Interpolation Backward Differentiation Formula
- LSODAR : Livermore Solver for Ordinary Differential equations with Automatic method switching for stiff and nonstiff problems, and with Root-finding
- MPI : Message Passing Interface
- NUMA : Non Uniform Memory Architecture
- OpenMP : Open Multi-Processing
- PEC : Prédicteur Evaluation Correcteur
- PLM : Product Lifecycle Management
- QSSA : Quasi Steady State Assumption
- RK : Runge Kutta
- SDIRK : Singly Diagonally Implicit Runge Kutta
- VC : Stratégie à coefficients variables
- VODE : Variable coefficient ODE solver
- VODPK : Variable coefficient ODE solver with Preconditionner Krylov
- WFR : WaveForm Relaxation

Chapitre 1

Définition du problème et état de l'art

Ce chapitre présente le problème que l'on cherche à résoudre ainsi que les méthodes classiquement utilisées. Les schémas numériques sont décrits et détaillés. Les définitions et les théorèmes habituels sont rappelés ainsi que les méthodes usuelles de parallélisation. De plus, le principal solveur EDO LSODAR est détaillé.

Sommaire

1.1 La résolution d'EDOs par les schémas numériques

Le problème dynamique continu que l'on cherche à résoudre est le problème EDO suivant :

$$\begin{cases} t \in [0, T], \\ \dfrac{\mathrm{d}\bar{y}}{\mathrm{d}t} = f(t, \bar{y}), \\ \bar{y}(0) = y_0, \end{cases} \tag{1.1}$$

avec $\bar{y} \in \mathbb{R}^d$. f est une fonction continue par rapport à t et $\bar{y}$ à valeurs dans $\mathbb{R}^d$. C'est un problème de dimension d pour lequel on pose l'hypothèse que la fonction f qui est très générale, linéaire ou non, est supposée continue.

Lorsque d'autres hypothèses sur f seront nécessaires, elles seront précisées.

1.1.1 Existence et unicité de la solution du problème initial

Le problème décrit ci-dessus apparaît dans de nombreux domaines tels que la physique, les sciences naturelles mais aussi les sciences sociales. Il n'est cependant pas possible pour la plupart de ces systèmes d'être résolus de manière continue exacte et on a donc recours à des méthodes d'approximation en utilisant des schémas numériques. Le problème (1.1) est appelé problème aux valeurs initiales ou **initial value problem** (IVP). Ce problème n'a cependant pas nécessairement une solution et cette solution n'est pas nécessairement unique, afin d'étudier le cas où la solution au problème existe et est unique on a besoin de se placer dans le cadre des hypothèses du théorème de Cauchy-Lipschitz (ou de Picard pour les anglophones).

À ce stade, on introduit l'hypothèse minimale sur les données de notre problème.

Hypothèse. *(H_{IVPS}) On considère un domaine ouvert $\Omega \subseteq \mathbb{R}^d$ et un intervalle ouvert $\mathbb{I} \subseteq \mathbb{R}$. On suppose que la fonction $f : \mathbb{I} \times \Omega \to \mathbb{R}^d$ est une fonction continue définie par $(t, \bar{y}) \mapsto f(t, \bar{y})$. On suppose également que $(t_0, y_0) \in \mathbb{I} \times \Omega$*

La durée d'existence d'une solution dépend du rectangle $R \subseteq \mathbb{I} \times \Omega$ centré sur (t_0, y_0), défini par deux réels a, b :

$$R = \{t : |t - t_0| \leq a\} \times \{\bar{y} : \|\bar{y} - \bar{y}_0\| \leq b\} \tag{1.2}$$

Comme $\mathbb{I} \times \Omega$ est domaine ouvert, il existe un tel rectangle R (pour certains réels a, b) pour tout couple $(t_0, y_0) \in \mathbb{I} \times \Omega$. On désigne par M, la valeur $M = \sup_R \|f(t, \bar{y})\|$.

De plus, on rappelle la définition de la continuité de Lipschitz :

Définition 1.1. (**Continuité de Lipschitz**) Une fonction f est dite Lipschitz continue sur un rectangle R par rapport à sa seconde variable $\bar{y}$ si il existe $K > 0$ tel que :

$$\|f(t,u) - f(t,v)\| \leq K\|u - v\| \quad \forall (t,u),(t,v) \in R. \tag{1.3}$$

Et dans ce cadre, on peut utiliser le théorème de Cauchy-Lipschitz qui nous permet de s'assurer que la solution du problème (1.1) existe et est unique :

Théorème 1.2. *(**Théorème de Cauchy-Lipschitz**) ([AP98]) On suppose l'hypothèse (H_{IVPS}) vérifiée. Soit f une fonction Lipschitz continue par rapport à sa seconde variable $\bar{y}$ sur R. Alors, l'IVP (1.1) a une solution unique sur l'intervalle $|t - t_0| \leq \delta$ avec $\delta = \min\{a, (1/K)\log(1 + (Kb/M))\}$*

Démonstration. La preuve de ce théorème est donnée par le théorème 12.1 de ([SM03]). □

On considère dans la suite du chapitre des méthodes numériques itératives dites "pas-à-pas" pour des solutions approchées du problème IVP (1.1). On suppose que la fonction f satisfait les conditions du théorème de Lipschitz. On considère l'IVP (1.1) résolue sur un intervalle $[t_0, T]$. Cet intervalle est subdivisé au moyen des points $t_n = t_0 + nh$, $n = 0, 1, \ldots, N$ avec $h = (T - t_0)/N$ et N un entier naturel positif. Le nombre réel h est appelé **pas de temps**. Pour chaque n, on cherche une solution numérique approchée y_n de $\bar{y}(t_n)$ la valeur analytique de la solution au temps t_n. Ces valeurs y_n sont calculées successivement pour $n = 1, 2, \ldots N$.

1.1.2 Les schémas numériques à un pas

Pour calculer de manière approchée le premier point y_1 de la solution, on dispose uniquement de la condition initiale et de la relation entre les dérivées des variables et les variables elles-mêmes du problème $\dot{y} = f(y)$. Les méthodes à un pas sont très utiles pour calculer le premier point de la solution à partir uniquement de la condition initiale. Nous verrons dans la Section 2.1 comment optimiser ces méthodes en termes de coût de calcul.

Généralités et erreurs

Une méthode dite "à un pas" exprime la solution y_{n+1} en utilisant y_n. La méthode la plus simple à un pas est la méthode d'Euler :

Définition 1.3. Méthode d'Euler Etant donné $\bar{y}(t_0) = y_0$, on définit :

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{1.4}$$

De manière plus générale, une méthode à "un pas " est écrite sous la forme :

$$y_{n+1} = y_n + h\phi(t_n, y_n, h), \quad n = 0, 1, \ldots, N-1, \quad \bar{y}(t_0) = y_0, \tag{1.5}$$

avec $\phi(\cdot,\cdot,\cdot)$ une fonction continue par rapport à toutes ses variables. Dans le cas d'Euler, on a $\phi(t_n, y_n, h) = f(t_n, y_n)$.

Afin de tester la précision des méthodes numériques, deux types d'erreurs sont définies : l'**erreur globale**, e_n telle que :

$$e_n = \bar{y}(t_n) - y_n, \tag{1.6}$$

et l'erreur de troncature :

$$T_n = \frac{\bar{y}(t_{n+1}) - \bar{y}(t_n)}{h} - \phi(t_n, \bar{y}(t_n), h). \tag{1.7}$$

Le théorème suivant relie l'erreur globale à l'erreur de troncature :

Théorème 1.4. *Soit une méthode générale à un pas (1.5), on suppose qu'en plus d'être continue par rapport à toutes ses variables, ϕ satisfait la condition de Lipschitz par rapport à la seconde variable sur un rectangle R avec une constante de Lipschitz L_ϕ. Alors en supposant que $\|y_n - y0\| \leq b$, $n = 1, 2, \ldots, N$, on a :*

$$\|e_n\| \leq \frac{T_r}{L_\phi}\left(e^{L_\phi(t_n - t_0)} - 1\right), \quad n = 0, 1, \ldots, N, \tag{1.8}$$

où $T_r = \max_{0 \leq n \leq N-1} \|T_n\|$.

Démonstration. La preuve de ce théorème se trouve dans le Chapitre 12 de [SM03]. □

Remarque : Le Théorème 1.4 donne un majorant de l'erreur globale qui est très pessimiste. En effet le terme en exponentielle explose quand le temps devient grand ou quand la constante de Lipschitz est grande. Dans cette thèse, on verra que pour certains cas ce majorant peut être amélioré (voir Section 2.2.5 du Chapitre 2)

Consistance et convergence

Le comportement des deux erreurs décrites dans la section précédente est analysé à l'aide de deux notions, la consistance d'une méthode numérique et la convergence d'une méthode numérique.

Définition de la **consistance** :

Définition 1.5. Le schéma numérique (1.5) est dit **consistant** par rapport à l'équation différentielle (1.1) si l'erreur de troncature (définie par (1.7)) est telle que quel que soit $\epsilon > 0$, il existe un réel positif $h(\epsilon)$ pour lequel $\|T_n\| < \epsilon$ pour $0 < h < h(\epsilon)$ et quel que soit les couples de points (t_n, y_n) et (t_{n+1}, y_{n+1}) de la courbe solution sur R.

La consistance est donc vue comme la convergence vers 0 de l'erreur de troncature quand le pas de temps h tend vers 0.

On montre que l'erreur de troncature pour le schéma d'Euler est :

$$\|T_n\| \leq Kh \quad pour \quad 0 < h \leq h_0, \tag{1.9}$$

avec K une constante indépendante de h.

La **convergence** illustre le fait que la solution approchée tend vers la solution exacte dans certaines circonstances qui sont illustrées dans le théorème suivant :

Théorème 1.6. *On suppose que le problème (1.1) vérifie le théorème de Cauchy-Lispchitz et que la solution approchée (1.5) reste dans la région R pour $h < h_0$. On suppose de plus que la fonction $\phi(\cdot, \cdot, \cdot)$ est continue par rapport à toutes ses variables sur le domaine $R \times [0, h_0]$ et est Lipschitz continue par rapport à sa deuxième variable. Alors si les approximations successives (y_n) générées en utilisant les temps $t_n = t_0 + nh$, $n = 1, 2, \ldots, N$*

sont obtenues à partir de (1.5) avec des pas de temps successifs h de plus en petits, chaque h étant plus petit que h_0, la solution numérique converge vers la solution exacte du problème dans le sens :

$$\lim_{n\to\infty} y_n = \bar{y}(t) \quad si \quad t_n \to t \in [t_0, T] \quad quand \quad h \to 0 \quad et \quad n \to \infty. \tag{1.10}$$

La convergence illustre le fait que l'erreur globale tende vers 0 quand h tend vers 0 (voir la démonstration du Théorème 12.3 de [SM03]).

Dans l'équation (1.10), l'erreur de troncature décroit avec une puissance de 1 par rapport à h quand h tend vers 0. Cependant, certains schémas améliorent la décroissance de l'erreur de troncature quand le pas de temps tend vers 0. Pour cela, on introduit l'**ordre de précision** d'un schéma numérique :

Définition 1.7. La méthode numérique (1.5) est dite d'**ordre** p si p est le plus grand entier positif tel que pour une solution suffisamment régulière $(t, \bar{y}(t))$ dans R du problème il existe une constante K et un pas de temps h_0 tel que :

$$\|T_n\| \leq Kh^p \quad pour \quad 0 < h \leq h_0. \tag{1.11}$$

Les schémas à un pas d'ordre supérieur à 1 font pour la plupart partie de la famille des méthodes de Runge-Kutta.

Les méthodes de Runge-Kutta

On a vu dans le paragraphe précédent que la méthode d'Euler est d'ordre 1, néanmoins il est possible d'obtenir des schémas à un pas d'ordre supérieur, ce sont les schémas de Runge-Kutta. Le principe de ces schémas est d'évaluer la fonction $f(\cdot,\cdot)$ en des points intermédiaires entre $(t_n, \bar{y}(t_n))$ et $(t_{n+1}, \bar{y}(t_{n+1})$. Plus précisément, on a la formulation suivante :

Définition 1.8. Un schéma de Runge-Kutta à s étapes intermédiaires s'écrit (voir [Kut01])

$$\begin{aligned} Y_i &= y_{n-1} + h\sum_{j=1}^{s} a_{ij} f(t_{n-1} + c_j h, Y_j), \qquad i = 1, \ldots, s, \\ y_n &= y_{n-1} + h\sum_{i=1}^{s} b_i f(t_{n-1} + c_i h, Y_i). \end{aligned}$$

Une définition équivalente est la suivante

Définition 1.9. Un schéma de Runge-Kutta à s étapes intermédiaires s'écrit également :

$$\begin{aligned} Y_i &= f\left(t_{n-1} + c_i h, y_{n-1} + h\sum_{j=1}^{s} a_{ij} Y_j\right), \qquad i = 1, \ldots, s, \\ y_n &= y_{n-1} + h\sum_{i=1}^{s} b_i Y_i. \end{aligned}$$

Afin de présenter de façon plus concise ces schémas, on présente les schémas de Runge-Kutta en donnant uniquement les valeurs des coefficients a_{ij}, c_i et b_i sous la forme d'un tableau appelé tableau de Butcher :

Définition 1.10. On définit le tableau de Butcher comme le tableau des coefficients (voir [But63]) :

$$\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1s} \\ c_2 & a_{21} & \dots & a_{2s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \dots & a_{ss} \\ \hline & b_1 & \dots & b_s \end{array}$$

Remarques : Les Y_i sont les approximations intermédiaires de la solution au temps $t_{n-1} + c_i h$. Les coefficients sont choisis de telle manière à ce que les différents termes d'erreurs s'annulent pour donner le bon ordre au schéma.

On donne ici les principaux schémas de Runge-Kutta explicites utilisés pour les ordres 2, 3 et 4 :

Ordre 2 :

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ \hline & 0 & 1 \end{array} \tag{1.12}$$

Ordre 3 :

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & 1/6 & 2/3 & 1/6 \end{array} \tag{1.13}$$

Ordre 4 :

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array} \tag{1.14}$$

Dans le cas des schémas de Runge-Kutta, le nombre d'évaluations de f est très important, et plus l'ordre est élevé plus ce nombre est grand. Cela peut devenir un problème pour les systèmes fortement couplés et de grande taille, pour lesquels le coût d'évaluation de f est grand. On s'est orienté alors vers des schémas d'ordre élevé qui nécessitent moins d'évaluations de f, les schémas multipas.

1.1.3 Les schémas numériques multipas

Les **schémas linéaires multipas** utilisent contrairement aux schémas à un pas *plusieurs* solutions issues des temps précédents, ainsi la forme générale d'un schéma linéaire multipas est

Définition 1.11. Un schéma linéaire multipas est :

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f(t_{n+j}, y_{n+j}), \tag{1.15}$$

où les coefficients $\alpha_0, \dots, \alpha_k$ et $\beta_0, \dots, \beta_k$ sont des constantes réelles (dans le cas où le pas de temps est fixe).

Il existe trois grandes familles de schémas multipas, les schémas d'Adams-Bashforth, Adams-Moulton et Backward Differentiation Formulae.

Les méthodes d'Adams-Bashforth sont explicites et ont la forme suivante :

$$\sum_{j=k-1}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k-1} \beta_j f(t_{n+j}, y_{n+j}). \tag{1.16}$$

Les méthodes d'Adams-Moulton sont implicites et ont la forme suivante :

$$\sum_{j=k-1}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f(t_{n+j}, y_{n+j}). \tag{1.17}$$

Les méthodes Backward Differentiation Formulae sont implicites et ont la forme suivante :

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\beta_k f(t_{n+k}, y_{n+k}). \tag{1.18}$$

Dans la Section 1.1.4 la propriété qui distingue ces trois types de schémas est donnée.

De même que pour les schémas à un pas il est important d'étudier le comportement du multipas et en particulier sa *stabilité*, sa *consistance* et sa *convergence*. Ces propriétés sont importantes car elles justifient l'utilisation d'un tel schéma. Dans la section 2.2 du Chapitre 2, ces propriétés permettent de construire de nouveaux schémas.

La zéro-stabilité

Le schéma numérique multipas (1.15) a besoin de k valeurs de départs qui sont $y_0, \ldots, y_{k-1}$, il est donc intéressant d'étudier l'influence des erreurs numériques de ces premiers points sur le comportement de la solution. On nomme alors cela la stabilité par rapport aux petites perturbations :

Définition 1.12. Une méthode linéaire à k-pas est dite **zero-stable** s'il existe une constante K telle que pour deux suites u_n et v_n générées par le même schéma mais en utilisant différentes valeurs initiales, $u_0, u_1, \ldots, u_{k-1}$ et $v_0, v_1, \ldots, v_{k-1}$ respectivement, on a

$$\|u_n - v_n\| \leq K \max\{\|u_0 - v_0\|, \|u_1 - v_1\|, \ldots, \|u_{k-1}, v_{k-1}\|\}, \tag{1.19}$$

pour $t_n \leq T$ et $h < h_0$.

On prouve que cela revient à étudier le comportement du schéma appliqué à l'équation triviale $y' = 0$. Cette propriété est en particulier utile dans la section 2.2.3 du Chapitre 2.

La consistance

De même que pour les schémas à un pas, il est important d'étudier la précision d'une méthode multipas. Pour cela, on introduit la notion d'erreur de troncature analogue à celle introduite pour les schémas à un pas :

$$T_n = \frac{\sum_{j=0}^{k} \left[\alpha_j \bar{y}(t_{n+j}) - h\beta_j f(t_{n+j}, \bar{y}(t_{n+j}))\right]}{h \sum_{j=0}^{k} \beta_j}, \tag{1.20}$$

et on a alors la définition :

Définition 1.13. Le schéma numérique (1.15) est dit consistant si l'erreur de troncature définie par (1.20) est telle que quel que soit $\epsilon > 0$ il existe $h(\epsilon)$ pour lequel

$$\|T_n\| < \epsilon \ \ \textit{pour}\ 0 < h < h(\epsilon). \tag{1.21}$$

De façon analogue aux schémas à un pas, on définit l'**ordre** d'une méthode multipas :

Définition 1.14. La méthode numérique (1.15) est dite d'**ordre** p, si p est le plus grand entier positif tel que pour une solution suffisamment régulière du problème IVP, il existe une constante K et un pas de temps h_0 tel que :

$$\|T_n\| \leq Kh^p \quad pour \quad 0 < h \leq h_0. \tag{1.22}$$

La convergence découle alors de la consistance et de la zéro-stabilité grâce au théorème d'équivalence de Dahlquist.

Théorème 1.15. *Pour une méthode multipas linéaire qui est consistante et où f est supposée satisfaire la condition de Lipschitz et avec des valeurs initiales consistantes (qui convergent vers y_0 quand h tend vers 0), la zero-stabilité est alors nécessaire et suffisante pour assurer la convergence. De plus, si la solution y a des dérivées d'ordre $p+1$ continues et une erreur de troncature en $\mathcal{O}(h^p)$ alors l'erreur globale de la méthode est aussi en $\mathcal{O}(h^p)$.*

Démonstration. La preuve est détaillée dans [Gau97] Théorème 6.3.4 ou [Hen62] Théorème 5.10. □

1.1.4 L'absolue stabilité

On a vu dans les deux parties précédentes les critères de convergence d'un schéma numérique à un pas ou multipas. Cette convergence est en particulier assurée lorsque le pas de temps h tend vers 0. Dans le cadre de l'accélération de la résolution des systèmes EDOs, il est important de choisir le pas de temps le plus grand possible (voir en particulier dans la section 1.1.6 du Chapitre 2 l'importance de ce choix). Ce choix du pas de temps est en grande partie restreint par la raideur du système qui impose un pas de temps maximal au schéma. Afin d'étudier ce phénomène, soit l'équation triviale

$$y' = \lambda y, \quad y(0) = y_0, \tag{1.23}$$

avec λ constant.

La solution de cette équation est $y(t) = \exp(\lambda t)$. Pour des valeurs négatives de λ, la solution est décroissante, on s'attend donc à ce que la solution numérique décroisse également. Si on utilise la méthode d'Euler explicite, on calcule la solution numérique $y_n^E = (1+h\lambda)^n y_0$, et de même en utilisant la méthode d'Euler implicite $y_n^I = (1-h\lambda)^{-n} y_0$. Ainsi quand $\lambda < 0$ et $h > 0$, on a $(1-h\lambda) > 1$, c'est pourquoi la suite ($|y_n^I|$ décroit de façon monotone quand n croît. Cependant pour $\lambda < 0$ et $h > 0$, $|1 + \lambda h| < 1$ si et seulement si $0 < h(-\lambda) < 2$. On a donc une limitation sur le pas de temps qui est $|\lambda|h < 2$ pour lequel on a bien une solution décroissante. Si cette condition n'est pas satisfaite alors la solution diverge.

On retrouve ce problème pour des systèmes linéaires d'EDOs, pour lesquels le pas de temps maximal est alors limité par la plus grande valeur propre de la matrice A du système $y' = Ay + B$. Le système est dit raide dans le cas où le ratio de la plus grande valeur propre et de la plus petite est très grand. En effet, dans ce cas, le plus grand pas de temps permis est exagérément petit par rapport à l'erreur commise.

Dans le cas des systèmes non linéaires, il n'est pas possible de définir la raideur comme précédemment, on utilise alors une linéarisation :

$$y'(t) = y'(t_n) + \frac{\partial f}{\partial t}(t_n, y(t_n))(t - t_n) + J(t_n)(y(t) - y(t_n)) + \dots \tag{1.24}$$

Une estimation de la raideur est le ratio de plus grande valeur propre et de la plus petite valeur propre de la matrice Jacobienne J.

On reprend l'exemple (1.23) et on considère que $\lambda \in \mathbb{C}$ avec $\mathrm{Re}(\lambda) < 0$. La solution de ce modèle converge dans $\mathbb{C}$ vers 0 quand $t \to \infty$. Ainsi une méthode linéaire multipas appliquée à cet exemple doit fournir une solution avec un pas de temps h qui doit se comporter de façon analogue. On définit alors l'absolue stabilité :

Définition 1.16. Une méthode linéaire multipas est dite absolument stable pour une valeur donnée de λh si on a $\lim_{n\to\infty} y_n = 0$ quand $t_n \to \infty$

Le but est de déterminer l'ensemble des valeurs λh pour lesquelles la méthode multipas est absolument stable. On donne alors la définition des régions d'absolue stabilité :

Définition 1.17. La région d'absolue stabilité d'une méthode multipas est l'ensemble des points λh du plan complexe pour lesquels la méthode est absolument stable.

Dans l'idéal, la zone d'absolue stabilité est le demi-plan complexe gauche entier, il n'y a alors pas de valeur limite pour h. Cette propriété s'appelle la A-stabilité :

Définition 1.18. Une méthode multipas est dite A-stable si la région d'absolue stabilité contient le demi-plan complexe gauche.

Dahlquist a énoncé deux propriétés concernant la A-stabilité :
- Aucune méthode explicite n'est A-stable.
- Aucune méthode multipas A-stable n'est d'ordre strictement supérieur à 2.

S'il n'est pas possible d'obtenir la A-stabilité on cherche à obtenir la plus grande zone de stabilité absolue possible. En particulier, on cherche à ce que l'axe des réels strictement négatifs soit entièrement compris dans cette zone. La famille de schémas vérifiant cette propriété est la famille des schémas Backward Differentiation Formulae (BDF). Les ordres 1 (au schéma d'Euler implicite) et 2 sont A-stables, les autres ont de très bonnes propriétés d'absolue stabilité (voir Figure 1.1).

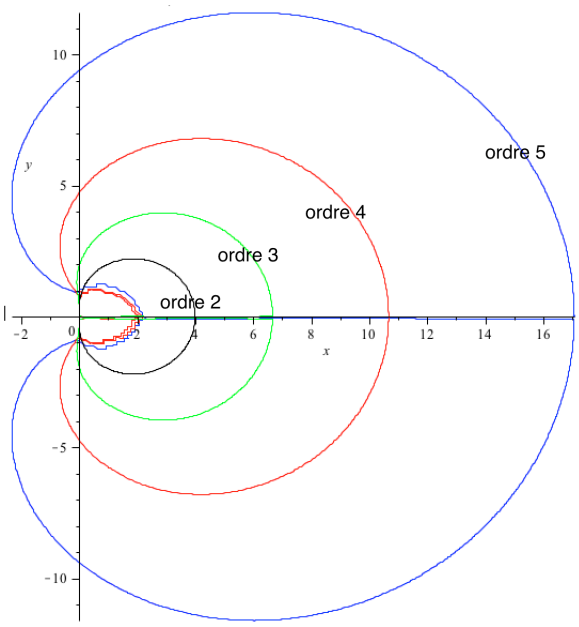


Figure 1.1 – Zones de stabilité pour le schéma BDF à l'extérieur des contours pour l'ordre 2, 3, 4 et 5.

A contrario, les schémas d'Adams-Bashforth et d'Adams-Moulton comme on peut le voir sur les Figures 1.2 et 1.3 respectivement ont des zones de stabilité très petites.

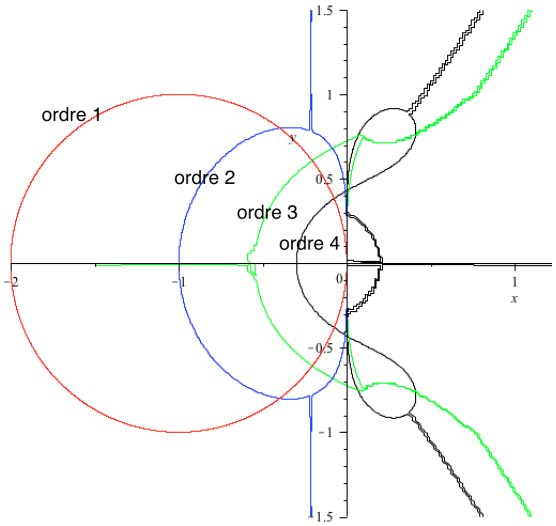


FIGURE 1.2 – Zones de stabilité pour le schéma Adams-Bashforth à l'intérieur des contours pour les ordres 1, 2, 3 et 4

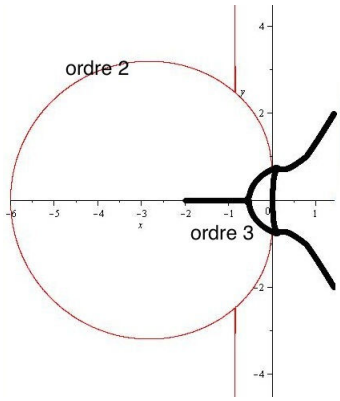


FIGURE 1.3 – Zones de stabilité pour le schéma Adams-Moulton à l'extérieur des contours pour les ordres 2 et 3

Une autre famille de schéma ayant de bonnes propriétés de A-stabilité sont les schémas Runge-Kutta implicites. Cependant, ces schémas sont coûteux, bien plus que les schémas BDFs. C'est pourquoi, ce sont les schémas BDFs qui serviront de modèle au développement d'un nouveau schéma plus performant encore. Ce schéma est développé dans la section 2.2 du Chapitre 2, et axé sur la résolution de la partie implicite du schéma (1.15). En effet, cette résolution est le plus souvent coûteuse ou alors peu stable.

1.1.5 Résolution des schémas implicites

On utilise classiquement trois méthodes pour la résolution des schémas linéaires multi-pas implicites de la forme (1.15) avec $\beta_0 \neq 0$. Pour les modèles non linéaires, on doit ainsi résoudre un système non linéaire de m équations.

La méthode du point fixe

La méthode la plus simple pour résoudre un tel système en y_n est la méthode du point fixe, c'est-à-dire en itérant sur l'expression suivante :

$$y_n^{\nu+1} = h\beta_0 f(t_n, y_n^{\nu}) - \sum_{j=1}^{k} \alpha_j y_{n-j} + h \sum_{j=1}^{k} \beta_j f_{n-j}, \tag{1.25}$$

avec ν le numéro de l'itération.

On a la convergence vers la solution y_{n+1} si $\|h\beta_0 \frac{\partial f}{\partial y}\| < 1$, ainsi cette méthode est appropriée pour des problèmes peu raides car la dérivée $\frac{\partial f}{\partial y}$ est grande pour les systèmes raides . On a vu dans la section précédente que le schéma BDF est intéressant car il est bien adapté aux systèmes raides cependant l'utilisation de la méthode point fixe impose une taille de pas de temps faible pour des systèmes raides afin de garantir la condition $\|h\beta_0 \frac{\partial f}{\partial y}\| < 1$. On perd donc tout l'intérêt de l'utilisation des schémas BDF. De plus, lorsque l'expression $\|h\beta_0 \frac{\partial f}{\partial y}\|$ approche 1, la convergence est très lente et il faut diminuer le pas de temps pour obtenir une convergence rapide. On comprend que la méthode de point fixe n'est donc pas adaptée aux systèmes raides que l'on cherche aussi à résoudre.

La méthode du prédicteur-correcteur

On a vu précédemment que la méthode du point-fixe ne converge pas pour les systèmes raides (à moins de prendre un pas de temps très petit) ; on utilise alors une méthode prédicteur-correcteur. Premièrement, y_n^0 est estimé à l'aide d'une méthode explicite de même ordre que la méthode implicite, ce point est alors appelé prédicteur (P) :

$$P : y_n^0 + \tilde{\alpha}_1 y_{n-1} + \ldots + \tilde{\alpha}_k y_{n-k} = h(\tilde{\beta}_1 f_{n-1} + \ldots + \tilde{\beta}_k f_{n-k}). \tag{1.26}$$

Ensuite on évalue la fonction f en y_n^0

$$E : f_n^0 = f(t_n, y_n^0). \tag{1.27}$$

Cette évaluation est insérée dans l'expression implicite dite *correcteur* pour obtenir une nouvelle évaluation de y_n que l'on nomme $y_n^{\nu+1}$ (ici $\nu = 0$) et on a :

$$C : y_n^{\nu+1} + \alpha_1 y_{n-1} + \ldots + \alpha_k y_{n-k} = h(\beta_0 f_0^{\nu} + \beta_1 f_{n-1} + \ldots + \beta_k f_{n-k}). \tag{1.28}$$

Si on arrête la procédure ici, on a alors une méthode dite PEC (Prédicteur Evaluation Correcteur). On peut également itérer un nombre ν fini de fois, on a alors une méthode dite $P(EC)^{\nu}$. Enfin, une évaluation finale permet de fournir la prochaine valeur de f_{n-1}. La méthode n'est pas très coûteuse mais comme on n'itère pas la méthode jusqu'à convergence, on ne conserve pas les propriétés de stabilité de la méthode dite "correcteur". La très bonne absolue stabilité de la méthode BDF est ainsi perdue, ou du moins en partie.

Méthode de Newton

La méthode de Newton est la méthode la plus largement utilisée. Elle s'écrit :

$$y_n^{\nu+1} = y_n^{\nu} - (I - h\beta_0 \frac{\partial f}{\partial y})^{-1} [\sum_{j=0}^{k} \alpha_j y_{n-j} - h \sum_{j=0}^{k} \beta_j f_{n-j}] \tag{1.29}$$

avec ν le numéro de l'itération, f_n et $\frac{\partial f}{\partial y}$ sont évaluées en y_n^{ν}. La valeur de y_n^0 est déterminée par interpolation (par exemple). Il est clair que cette méthode est extrêmement coûteuse

car, en théorie, il faut évaluer la Jacobienne à chaque itération. En pratique les solveurs usuels essaient de limiter le nombre d'évaluations. Toutefois, pour des problèmes fortement non linéaires, l'évaluation régulière de la Jacobienne est nécessaire pour chaque itération, et dans le cas où ces problèmes non linéaires sont de grande taille, le coût de cette évaluation devient très élevé.

Il n'existe donc pas de méthode satisfaisante pour la résolution de la partie implicite des schémas numériques. En effet, soit les propriétés de stabilité sont perdues, soit le coût de résolution est très élevé. Le nouveau schéma numérique proposé dans cette thèse et développé dans la Section 2.2 du Chapitre 2 s'attache à résoudre ce problème. On a cherché en particulier à conserver les propriétés de stabilité du schéma BDF à l'identique tout en accélérant la résolution de la partie implicite.

1.1.6 Estimation de l'erreur et adaptation du pas de temps

Les schémas numériques à pas de temps variable (voir aussi Section 2.3.3 du Chapitre 2) permettent un meilleur contrôle de l'erreur commise par rapport à une tolérance limite fixée par l'utilisateur. On parle alors de **contrôle de l'erreur**. Lorsque l'utilisateur d'un solveur EDO souhaite contrôler l'erreur commise soit à chaque pas de temps (erreur locale), soit à la fin de la simulation (erreur globale), il est important de connaître les différentes erreurs commises par le schéma numérique. Cependant, l'évaluation exacte de cette erreur est souvent impossible car la solution continue exacte du problème est inconnue. On a alors recours à des estimateurs. On parle d'**estimation de l'erreur**. Le paragraphe suivant détaille les principaux estimateurs et les méthodes de contrôle de l'erreur globale/locale. Dans la section 2.3.1 du Chapitre 2, ces estimateurs sont adaptés à notre méthode afin de minimiser leur coût de calcul.

Motivations

Estimer l'erreur assure à la solution numérique approchée une certaine cohérence vis-à-vis de la solution exacte. Cependant, l'estimation de cette erreur coûte cher et il est important (cf section 2.3.1 du Chapitre 2) d'adapter l'estimateur à la méthode numérique utilisée. En pratique, l'utilisateur fixe une tolérance τ et le schéma numérique utilise alors un pas de temps qui donne une mesure d'erreur inférieure à τ. Afin de gagner en rapidité, on choisit alors un pas de temps le plus grand possible tout en respectant cette tolérance. On a vu dans les différents paragraphes précédents que le pas de temps peut être limité suivant le schéma utilisé afin de garantir la stabilité du schéma. C'est en particulier vrai pour les systèmes raides pour lesquels le pas de temps doit être choisi petit par rapport à un système moins raide. On a également vu dans la Section 1.1.4 que certaines méthodes (les méthodes BDF) permettaient de lever cette limite de pas de temps.

L'erreur globale

L'erreur globale (voir Section 1.1.2) est définie par

$$\bar{y}(t_n) - y_n. \tag{1.30}$$

Cependant un solveur n'estime habituellement pas cette erreur et de plus il n'est souvent pas possible de contrôler cette erreur. C'est la stabilité du schéma qui représente la propagation des erreurs numériques commises à chaque pas.

L'erreur locale

Comme vu dans la Section 1.1.2, l'erreur locale (ou erreur de troncature) est définie en utilisant $u(t)$ la solution du problème

$$u' = f(t, u), \tag{1.31}$$

$$u(t_n) = y_n, \tag{1.32}$$

et l'erreur locale est alors

$$le_n = \frac{u(t_n + h_n) - y_{n+1}}{h}. \tag{1.33}$$

Cette erreur mesure comment se comporte la méthode numérique sur un pas. Pour une méthode d'ordre p, on rappelle qu'on a

$$le_n = h_n^p \phi(t_n, y_n) + O(h_n^{p+1}). \tag{1.34}$$

Un estimateur classique consiste à utiliser deux formules d'intégration pour un pas. On considère par exemple une approximation y_{n+1} d'ordre p et une autre approximation y^*_{n+1} d'ordre $p* > p$. Alors on a

$$est_n = y^*_{n+1} - y_{n+1} \qquad = [u(t_n + h_n) - y_{n+1}] - [u(t_n + h_n) - y^*_{n+1}] \tag{1.35}$$

$$= le_n + O(h^{p+1}). \tag{1.36}$$

On remarque que est_n est un estimateur de l'erreur locale de la formule d'ordre le plus faible. La plupart des codes qui utilisent ce type d'estimateur poursuivent l'intégration avec la formule d'ordre plus élevé (on suppose que cette formule est plus précise). est_n est donc un estimateur pessimiste par rapport à l'erreur locale réellement commise. On verra dans la section 2.3.1 du Chapitre 2 comment minimiser le coût de calcul de cet estimateur et comment ajuster au mieux l'estimateur par rapport à l'erreur locale.

Contrôle du pas de temps

On estime l'erreur pour deux principales raisons :
- donner de la valeur à la solution approchée en montrant qu'elle diffère de la solution exacte par une erreur plus faible que la tolérance fixée par l'utilisateur,
- grandir le pas de temps le plus possible (et ainsi accélérer la résolution) tout en respectant la tolérance de l'utilisateur.

On rappelle l'expression de l'estimateur :

$$est_n = le_n + h.o.t., \tag{1.37}$$

où *h.o.t.* désigne les termes d'ordres plus élevés (higher order terms). Si on désigne par τ la tolérance et par $\|\cdot\|$ une norme, on doit avoir

$$\|est_n\| \leq \tau. \tag{1.38}$$

Si l'estimateur ne respecte pas cette inégalité, alors la solution sur ce pas de temps est rejetée et recalculée avec un pas de temps plus petit. À partir de l'expression de l'erreur locale (1.34), on suppose que si le pas de temps est réduit de h_n à αh_n (α un réel compris entre 0 et 1), alors l'erreur locale sera réduite de le_n à $\alpha^p le_n$. Cela indique le plus grand α pour lequel l'estimateur satisfait la condition (1.38) qui est :

$$\alpha = \left(\frac{\tau}{\|est_n\|}\right)^{1/p}. \tag{1.39}$$

De même, si l'estimateur est trop petit par rapport à la tolérance, on choisit alors d'augmenter le pas de temps de h_n vers αh_n (α un réel plus grand que 1). Le plus grand α qui vérifie l'inégalité (1.38) est alors le même que (1.39).

1.1.7 Les solveurs EDOs

Il existe de nombreux solveurs EDOs. Ils se distinguent par rapport au type de problème qu'ils traitent et pour lesquels ils sont le plus efficace. Il est en effet très difficile d'implémenter un solveur EDO qui soit polyvalent et capable de résoudre n'importe quel problème de manière efficace. Cette thèse est centrée sur les problèmes de grande taille, non linéaires et raides. Ces problèmes regroupent les principales difficultés rencontrées par les solveurs EDOs. Le Lawrence Livermore National Laboratory a développé un package de solveurs pour les problèmes EDOs dits IVP (c'est-à-dire sans équations algébriques). Ce package regroupe différents solveurs avec différentes propriétés :

- LSODE dont les propriétés sont détaillées ci-dessous
- VODE identique à LSODE mais qui utilise des coefficients variables (en fonction du pas de temps) au lieu d'interpolations (la différence entre ces deux méthodes est développée dans la Section 2.3.3 du Chapitre 2)
- VODPK identique à VODE mais qui utilise un préconditionneur de Krylov afin de résoudre les systèmes EDOs implicites.

Intéressons-nous maintenant plus en détail au solveur le plus largement utilisé parmi les logiciels de calcul numérique : LSODE. LSODE est un solveur de référence pour les EDOs. Il est développé par le Lawrence Livermore National Laboratory depuis 1993. Ce paragraphe détaille les principales caractéristiques de ce solveur et en particulier les schémas utilisés.

Le solveur LSODE utilise principalement deux méthodes d'intégration suivant la raideur du problème. Une méthode d'Adams-Moulton à pas variable et à ordre variable entre 1 et 12 et une méthode de type BDF à pas variable et à ordre variable entre 1 et 5. Ces deux méthodes utilisent une méthode de type Prédicteur-Correcteur et ensuite différentes options permettent de choisir la méthode d'itération sur le correcteur :

- option 0 : itération de type point-fixe
- option 1 : itération de type Newton modifié avec une Jacobienne fournie par l'utilisateur
- option 2 : itération de type Newton modifié avec une Jacobienne estimée numériquement
- option 3 : itération de type Jacobi-Newton avec une Jacobienne estimée numériquement

À chaque pas de temps LSODE stocke une matrice appelée matrice de Nordsieck. Cette matrice contient les estimations des dérivées q-ièmes de la solution pour q variant de 1 à p avec p l'ordre du schéma. Cette matrice permet deux choses :

- L'initialisation du prédicteur en utilisant un développement de Taylor à l'ordre correspondant.
- L'estimation de l'erreur locale en estimant directement le terme de troncature.

On a vu dans les sections précédentes que le coût de calcul des itérations de Newton est très important dans le cas des systèmes de grande taille ou dans le cas des systèmes fortement non linéaires. Le Chapitre 2 s'applique donc à proposer une nouvelle méthode de résolution des systèmes EDOs qui soit à la fois stable et rapide. On verra également dans la section 2.3.1 du Chapitre 2 comment estimer l'erreur locale en évitant d'estimer les différentes dérivées.

1.1.8 La méthode WFR

On introduit ici la méthode de WaveForm Relaxation (acronyme WFR). La méthode Waveform Relaxation a été introduite dans [aRASV82]. Il s'agit d'un processus origi-

naire du théorème de Picard, qui rend possible la résolution simultanée et en parallèle de sous-systèmes couplés sur des fenêtres de simulation successives. Chaque sous-système est caractérisé par sa waveform (c'est-à-dire sa solution sur un intervalle de temps déterminé). Le but est de déterminer la waveforme d'un sous-système, en considérant toutes les waveforms des autres sous-systèmes constantes pendant une itération. La méthode WFR est adaptée à la résolution des systèmes de grande taille creux afin de diviser le système complet en plusieurs sous-systèmes qui seront résolus sur différents processeurs. Cependant la méthode WFR est également vue comme une méthode numérique utilisant un seul processeur et en travaillant sur plusieurs parties d'un même processeur. L'utilisation de la WFR en multiprocesseurs est plus largement détaillée dans la section 2.5 du Chapitre 2. La méthode WFR est appliquée sur des schémas classiques tels que Euler implicite/explicite et il est important d'étudier les critères de convergence de cette méthode pour les schémas numériques. En effet, la raideur et le pas de temps modifient le comportement de la méthode WFR et peuvent la faire diverger. Cette convergence est étudiée dans la Section 2.5 du Chapitre 2

On illustre maintenant la méthode WFR avec un système à 3 variables.

Application de l'algorithme WFR sur un exemple de taille 3

On considère un système EDO à trois variables d'état x, y, z. On note x_k^m la solution approchée de la solution exacte à t_k et pour l'itération m. L'intégration du système EDO est faite sur trois processeurs ayant chacun leur propre mémoire.

A $t = 0$ les trois processeurs ont dans leur mémoire toutes les conditions initiales du problème à savoir x_0^1, y_0^1, z_0^1. Le principe de calcul est le suivant :

- Le processeur 1 intègre la variable x et calcule une valeur approchée de $x_1^1, x_2^1, .., x_f^1$ les valeurs des variables y et z restent constantes et égales à y_0^1, z_0^1
- Le processeur 2 intègre la variable y et calcule une valeur approchée de $y_1^1, y_2^1, .., y_f^1$ les valeurs des variables x et z restent constantes et égales à x_0^1, z_0^1
- Le processeur 3 intègre la variable z et calcule une valeur approchée de $z_1^1, z_2^1, .., z_f^1$ les valeurs des variables x et y restent constantes et égales à x_0^1, y_0^1

A la fin de l'intégration tous les processeurs échangent les valeurs de leurs variables respectives pour tous les instants et le processus est réitéré :

- Le processeur 1 intègre la variable x et calcule une valeur approchée de $x_1^2, x_2^2, .., x_f^2$ les valeurs des variables y et z restent égales à, respectivement $y_1^1, y_2^1, .., y_f^1$ et $z_1^1, z_2^1, .., z_f^1$.
- Le processeur 2 intègre la variable y et calcule une valeur approchée de $y_1^2, y_2^2, .., y_f^2$ les valeurs des variables x et z restent égales à, respectivement $x_1^1, x_2^1, .., x_f^1$ et $z_1^1, z_2^1, .., z_f^1$.
- Le processeur 3 intègre la variable z et calcule une valeur approchée de $z_1^2, z_2^2, .., z_f^2$ les valeurs des variables x et y restent égales à, respectivement $x_1^1, x_2^1, .., x_f^1$ et $y_1^1, y_2^1, .., y_f^1$.

Ce processus est réitéré jusqu'à convergence de toutes les variables.

La méthode WFR est présentée plus en détail dans la partie 2.5 du Chapitre 2. On applique en particulier la méthode aux schémas numériques classiques et on étudie leur convergence.

1.2 Les méthodes de parallélisation

Dans cette section, on détaille les différentes architectures de calcul parallèle et les méthodes de parallélisation associées. Cette parallélisation est particulièrement utile afin de répartir la charge de calcul d'un système de grande taille sur les différents moyens de calcul qui seront à diposition. Ainsi on pourra accélérer la résolution d'un grand système EDO en calculant simultanément les solutions correspondantes à plusieurs sous-systèmes issus du système complet.

1.2.1 La parallélisation en mémoire partagée

La parallélisation en mémoire partagée est le premier type de parallélisation que l'on a mis en place.

Les architectures de type mémoire partagée

Dans une architecture de type mémoire partagée, les différents processus de calcul ont accès à une même zone de la mémoire vive. Les différents processus sont alors appelés threads. Typiquement, ce genre d'architecture correspond aux architectures implantées dans les ordinateurs actuels. Un processeur possède plusieurs coeurs qui vont correspondre chacun à un processus (ou thread). Les différents coeurs ont accès à la même mémoire [Bar], voir Figure 1.4.

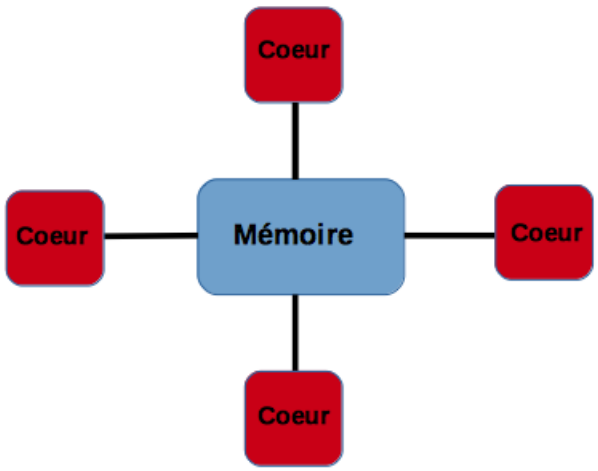


FIGURE 1.4 – Architecture de type mémoire partagée

Les avantages des architectures de type mémoire partagée sont nombreux. Tout d'abord, il n'y a pas de communication entre les différents processus de calcul car ils ont tous accès à la même mémoire. Les données nécessaires pour un processus sont donc mises à jour par tous les autres proccesus sans communication entre les coeurs. Le second avantage est une conséquence directe du premier : la parallélisation d'un code en mémoire partagée est donc aisée car l'utilisateur n'a pas à construire des communications entre les threads [Bar].

Cette mémoire commune présente cependant quelques inconvénients.

– Le premier est le risque de goulot d'étranglement. En effet, toutes les données provenant des différents coeurs transitent dans une seule mémoire. Dans le cas où le nombre de coeurs est grand, la vitesse d'exécution va être conditionnée par la vitesse à laquelle circulent les données entre les coeurs et la mémoire dans un bus unique.
– Un autre problème des architectures à mémoire partagée est la cohérence du cache. En effet si un processeur travaille sur certaines données, il faut s'assurer que ses données sont à jour et que deux processus ne travaillent pas simultanément sur les mêmes données.

Il existe un troisième inconvénient de ce type d'architecture : les zones de mémoire peuvent être plus ou moins éloignées des coeurs de calcul ce qui peut engendrer des différences de vitesse de calcul selon les coeurs. On appelle ce genre d'architecture une architecture NUMA (Non Uniform Memory Architecture). Enfin, le dernier inconvénient est qu'il est difficile d'accroître le nombre de coeurs partageant la même mémoire [Bar]. Aujourd'hui le maximum de coeurs pour une seule mémoire est de 16.

Les directives OpenMP

La principale méthode de parallélisation appliquée aux architectures en mémoire partagée utilise les directives OpenMP. Dans le paragraphe suivant, on donne un résumé succinct de ces directives en détaillant leur rôle et leur mise en place. La section 3.3 du Chapitre 3 détaille plus particulièrement certaines directives utilisées dans le code. OpenMP est un ensemble de procédures et de directives de compilation qui ont pour but de diminuer le temps d'exécution d'un programme sans changer sa sémantique.

Principaux concepts

Le programme OpenMP est exécuté par un seul processus qui active des processus légers (appelés threads) à l'entrée d'une région parallèle. Chaque processus léger exécute ensuite une tâche composée d'un ensemble d'instructions. Lors de l'exécution d'une tâche, une variable peut être soit :

– **variable privée**, elle est stockée dans la mémoire locale d'un processus léger,
– **variable partagée**, elle est stockée dans un espace mémoire partagé par tous les processus légers.

Le programme OpenMP présente une alternance de régions séquentielles et de régions parallèles. Les régions séquentielles sont exécutées par la tâche maître, numéro 0. Les régions parallèles peuvent être exécutées par plusieurs tâches à la fois et les tâches peuvent se partager le travail dans les régions parallèles.

Le travail se partage selon plusieurs possibilités :

– Les tâches se répartissent les itérations d'une boucle (ce partage est principalement utilisé par notre code) ;
– Les tâches exécutent différentes sections de code mais une seule section par tâche ;
– Les tâches exécutent plusieurs occurrences d'une même procédure.

Une synchronisation est introduite entre les différentes tâches afin d'éviter une mise à jour de variable partagée par plusieurs tâches. Le système d'exploitation affecte les différents coeurs de calcul aux tâches. Dans l'idéal il y a un coeur de calcul par tâche.

Structures d'OpenMP

OpenMP se structure suivant les différents éléments suivants :

– **Les directives** définissent le partage du travail, la synchronisation et le statut privé ou partagé des variables
– **Les fonctions** qui appartiennent à une bibliothèque éditant les liens du programme
– **Les variables d'environnement** dont les valeurs sont prises en compte à l'exécution.

Lors de la compilation, il est nécessaire d'utiliser des options de compilation :

– -openmp pour les compilateurs Intel,
– -fopenmp pour les compilateurs gcc.

Le nombre de tâches souhaitées doit également être indiqué lors de la compilation avec la commande :

```
export OMP_NUM_THREADS=4
```

Le détail des différentes directives OpenMP est détaillé dans l'Annexe A

Performances

La performance dépend beaucoup de l'architecture de la machine mais certaines règles permettent d'optimiser l'accélération du code OpenMP par rapport à son exécution séquentielle.

– La principale règle est de minimiser le nombre de régions parallèles dans le code et d'adapter le nombre de tâches à la taille du problème.
– Il est toujours mieux de paralléliser les boucles les plus externes.
– La parallélisation de certaines boucles n'est utile qu'à partir d'une certaine taille. La clause IF permet de ne paralléliser la boucle qu'au delà d'une certaine taille.

1.2.2 La parallélisation en mémoire distribuée

La parallélisation en mémoire distribuée est moins évidente et moins facile à mettre en place que la parallélisation en mémoire partagée. Ce paragraphe détaille le type d'architecture utilisée ainsi que la bibliothèque de fonctions permettant la parallélisation sur ce type d'architecture.

L'architecture en mémoire distribuée

Contrairement aux architectures de type mémoire partagée, la mémoire dans les architectures à mémoire distribuée n'est plus commune à tous les processus de calcul. Chaque noeud de calcul possède donc sa propre mémoire mais est capable d'échanger des données avec la mémoire des autres noeuds de calcul [Bar], comme on peut le voir sur la Figure 1.5.

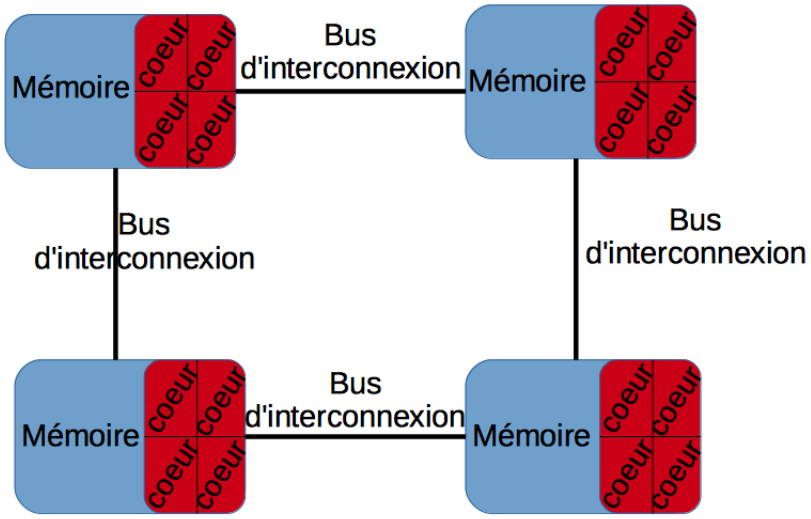


Figure 1.5 – Architecture de type mémoire distribuée

L'avantage majeur de ce type d'architecture est de pouvoir augmenter le nombre de noeuds de calcul et ainsi de développer des supercalculateurs avec de très nombreux noeuds identiques. Les capacités de calcul de ces supercalculateurs sont bien supérieures à celles des machines en mémoire partagée.

Ce type d'architecture présente quelques inconvénients. Chaque processus travaillant avec des données stockées sur sa propre mémoire, il doit être capable de recevoir d'autres données et d'envoyer ses propres données. Ces communications entre les différents processus sont coûteuses et, de plus, ces communications doivent être spécifiées dans le code par le programmeur, ce qui complexifie l'implémentation.

1.2.3 La bibliothèque MPI

Le MPI est une bibliothèque de fonctions permettant d'échanger des données entre les différents noeuds de calcul d'une architecture à mémoire distribuée. Ces fonctions sont utilisables en C ou en Fortran et sont très utilisées dans les calculs sur les clusters [GLS96].

Concepts

Toutes les variables du programme sont privées et sont stockées dans la mémoire locale de chaque processus. Les processus exécutent éventuellement des parties différentes du code et les données sont échangées en utilisant des fonctions MPI. Chaque processus est numéroté par une variable que l'on appelle rang.

Chaque message envoyé par un processus doit être reçu par un autre processus. Le message est constitué de données mais également de l'identificateur du processus émetteur, du type de donnée, de sa taille et de l'identificateur du processus récepteur.

L'architecture des supercalculateurs

La mémoire dans les supercalculateurs est distribuée selon les différents noeuds de calculs. Dans chaque noeud de calcul, la mémoire est partagée et plusieurs unités de calcul peuvent être présentes.

Historique

V1.0 : en 1994, une quarantaine d'organisations définissent un ensemble de sous-programmes qui constitue la bibliothèque d'échanges de messages.
V2.0 : Apparition des entrées/sorties parallèles, de la copie de mémoire à mémoire.
V3.0 : en 2012, ajout des communications collectives non bloquantes
Il existe des implémentations MPI opensource dont open-mpi.

Le détail des fonctions MPI est donné dans l'Annexe B.

1.3 L'importation des modèles depuis Modelica

Un langage de modélisation trés intéressant est le langage Modelica. Ce langage est très utilisé par les industriels afin de modéliser des systèmes complexes en utilisant en particulier la bibliothèque de modèles élémentaires. Plusieurs logiciels permettent cette modélisation en langage Modelica et contiennent différents solveurs permettant de résoudre ces modèles. Ces logiciels permettent également d'exporter les modèles industriels afin qu'ils puissent être résolus au moyen d'autres solveurs et intégrés dans d'autres logiciels.

1.3.1 Le langage de modélisation Modelica

La thèse porte sur la résolution des modèles issus de Modelica et non pas sur le langage Modelica. Cependant, les propriétés principales de ce langage sont décrites dans les lignes suivantes.

Modelica est un langage de modélisation déclaratif, orienté objet et multi-domaine. Des modèles mécaniques, électriques, électroniques, hydrauliques, thermiques entre autres peuvent être ainsi construits avec ce langage. Ce langage est libre et est géré par l'association Modelica. Cette association a également développé une librairie standard qui regroupe plus de 1300 composants de base dans tous les domaines de la physique.

Principales caractéristiques

Modelica est orienté objet comme Java ou C++ mais est avant tout un langage de modélisation se focalisant sur des objets qui se comportent comme des systèmes de simulation. De plus, ce langage est acausal, c'est-à-dire qu'une équation n'est pas une assignation de la partie droite vers la partie gauche, mais les deux parties gauche et droite sont manipulées de manière symbolique. Certaines parties de code Modelica seront détaillées dans le Chapitre 4 consacré aux applications.

Développement

La version 1.0 de ce langage fut développée par Hilding Elmqvist. Par la suite, différentes évolutions ont progressivement permis la modélisation de modèles de plus en plus variés. La version actuelle est la 3.3. La librairie standard n'a de cesse de se développer de manière complètement libre avec de plus en plus d'éléments accessibles par tous.

1.3.2 Les logiciels de modélisation utilisant Modelica

Il existe deux familles de logiciels dans lesquels le langage Modelica est mis en oeuvre, les logiciels commerciaux tout d'abord :

– CATIA commercialisé par Dassault Systèmes, via le noyau Dymola principal intégrateur commercial de Modelica,
– LMS Imagine via AMESim développé par LMS International,
– MapleSim qui inclut le noyau de calcul formel Maple, développé par Maplesoft,
– MathModelica de Wolfram Research,
– Simulation X de ITI GmbH.

ll y a ensuite les logiciels libres :
– JModelica.org de l'Université de Lund avec Modelon AB,
– OpenModelica de l'Université de Linköping,
– Modelicac de Scilab-Xcos/Scicos.

Il existe également beaucoup d'autres logiciels capables d'exécuter des modèles issus de Modelica, via l'échange de modèles.

1.3.3 L'échange de modèles

On a vu dans le paragraphe précédent que le nombre de logiciels utilisant le langage Modelica est relativement restreint. La question est donc de savoir si d'autres logiciels utilisant d'autres langages peuvent exécuter des modèles créés en Modelica. Pour cela, on définit un standard d'échange de modèle, le FMI (Functionnal Mock-up Interface), qui est une interface standardisée utilisée pour la simulation de modèles multiphysiques. Ce standard ainsi que l'échange de modèles sont plus amplement décrits dans les thèses de [BKEF14] et [HA08].

Origines

Le standard FMI a été développé originellement par un consortium européen d'industriels mené par Dassault Systèmes sous le nom de MODELISAR.

Idées

La définition d'un standard FMI a pour but de permettre quatre choses :
– l'échange de modèles entre différents outils de simulation. On peut ainsi construire un modèle complexe en utilisant différents composants créés par différents outils de simulation et simuler ce modèle complexe à l'aide d'un solveur. Cette fonctionnalité est appelée : Model Exchange,
– la co-simulation de modèles issus de différents outils de simulation. Chaque modèle est simulé individuellement avec son propre solveur. Cette fonctionnalité est présente dans le standard 2.0. ,
– l'utilisation dans des applications industrielles AUTOSAR,
– l'utilisation pour la PLM (Product Lifecycle Management, modèles et données relatives au management de cycle de vie de produit).

En pratique, l'implémentation de l'interface FMI dans un outil de simulation permet la création de modèles de simulation interconnectés.

Fonctionnement

Le fonctionnement du standard FMI est le suivant : un outil de simulation décrit un modèle et ses équations qui sont ensuite encapsulés et exportés dans FMU (Functionnal Mock-up Unit). Un autre outil de simulation peut alors importer ce FMU et l'exécuter avec d'autres FMUs.

Contenu du fichier

Le FMU est un fichier .fmu de type zip qui contient plusieurs fichiers :
- un fichier .xml qui contient la description du modèle et la définition des variables,
- les équations du modèle contenues dans les fichiers,
- des données optionnelles telles que la documentation associée au modèle.

Outils de simulation supportant FMU

Les deux tableaux suivants Table 1.1 et Table 1.2 donnent l'ensemble des logiciels utilisant le standard FMU ,soit en Model Exchange, soit en CoSimulation (ou les deux).

		Model Exchange		Cosimulation	
Tools	FMI Version	Export	Import	Slave	Master
Adams	FMI 1.0			✓	✓
Amesim	FMI 1.0	✓	✓	✓	✓
ANSYS SCADE Display	FMI 1.0	✓		✓	
ANSYS SCADE Suite	FMI 1.0	✓		✓	
ANSYS Simplorer	FMI 1.0		✓		
ASim - Autosar Simulation	FMI 1.0	✓		✓	
@Source	FMI 1.0	✓			
AVL CRUISE	FMI 1.0		✓	✓	✓
Building Controls Virtual Test Bed	FMI 1.0				✓
CarMaker	FMI 1.0				✓
CATIA	FMI 1.0	✓	✓	✓	✓
ControlBuild	FMI 1.0	✓	✓	✓	✓
CosiMate	FMI 1.0	✓	✓	✓	✓
Cybernetica CENIT	FMI 1.0		✓		
Cybernetica ModelFit	FMI 1.0		✓		✓
DSHplus	FMI 1.0			✓	
dSPACE SCALEXIO	FMI 1.0				✓
dSPACE SYNECT	FMI 1.0				
dSPACE VEOS	FMI 1.0				✓
Dymola	FMI 1.0	✓	✓	✓	✓
EnergyPlus	FMI 1.0			✓	✓
ETAS-ASCMO	FMI 1.0			✓	
ETAS-FMI-based Integration and Simulation Platform	FMI 1.0				
ETAS-FMU Generator for ASCET	FMI 1.0				
ETAS-FMU Generator for Simulink	FMI 1.0				
ETAS-INCA-FLOW (MiL/SiL Connector)	FMI 1.0				✓
ETAS-ISOLAR-EVE (ETAS Virtual ECU)	FMI 1.0			✓	
ETAS-LABCAR-OPERATOR	FMI 1.0				✓
Flowmaster	FMI 1.0	✓			
FMI Add-in for Excel	FMI 1.0				✓
FMI add-on for NI VeriStand	FMI 1.0		✓		✓
FMI Blockset for Simulink	FMI 1.0				✓
FMI Library	FMI 1.0		✓		✓

Table 1.1 – Logiciels utilisant le standard FMU (1ère partie)

		Model Exchange		Cosimulation	
Tools	FMI Version	Export	Import	Slave	Master
FMI Target for Simulink Coder	FMI 1.0			✓	
FMI Toolbox for Carmaker	FMI 1.0		✓		✓
FMI Toolbox for MATLAB/Simulink	FMI 1.0	✓	✓	✓	✓
FMUSDK	FMI 1.0	✓	✓	✓	✓
GT-SUITE	FMI 1.0	✓	✓	✓	✓
Hopsan	FMI 1.0	✓	✓		
IBM Rational Rhapsody	FMI 1.0	✓			
ICOS "Independent Co-Simulation	FMI 1.0		✓	✓	✓
JavaFMI	FMI 1.0				✓
JFMI	FMI 1.0			✓	✓
JModelica.org	FMI 1.0	✓	✓	✓	✓
LMS Virtual.Lab Motion	FMI 1.0		✓	✓	✓
MapleSim	FMI 1.0	✓			
MESSINA	FMI 1.0		✓		✓
MWorks	FMI 1.0	✓			
NI LabView	FMI 1.0				
OpenModelica	FMI 1.0	✓	✓		✓
OPTIMICA Studio	FMI 1.0	✓	✓		✓
Ptolemy II	FMI 1.0				
PyFMI	FMI 1.0		✓		✓
RecurDyn	FMI 1.0				✓
Reference FMUs	FMI 1.0				
Silver	FMI 1.0	✓	✓	✓	✓
SIMPACK	FMI 1.0		✓	✓	✓
SimulationX	FMI 1.0	✓	✓	✓	✓
SystemModeler	FMI 1.0	✓			
TLK FMI Suite	FMI 1.0		✓		✓
TLK TISC Suite	FMI 1.0		✓		✓
TWT Co-Simulation Framework	FMI 1.0			✓	✓
TWT FMU Trust Centre	FMI 1.0			✓	
xMOD	FMI 1.0		✓		✓

TABLE 1.2 – Logiciels utilisant le standard FMU (2ème partie)

Chapitre 2

Accélération par l'utilisation de nouveaux schémas numériques

Dans ce chapitre, on s'intéresse plus particulièrement aux problèmes aux valeurs initiales impliquant des systèmes raides et de grande taille d'équations différentielles ordinaires définis en 1.1 mais sous la forme autonome (la fonction f ne dépend plus que de la solution $\bar{y}$) :

$$\begin{cases} t \in [0, T], \\ \dfrac{\mathrm{d}\bar{y}}{\mathrm{d}t} = f(\bar{y}), \\ \bar{y}(0) = y_0, \end{cases} \tag{2.1}$$

y est un vecteur de $\mathbb{R}^d$, f est une fonction continue sur $\mathbb{R}^d$ et à valeurs dans $\mathbb{R}^d$.

Les systèmes non linéaires, raides et de grande taille nécessitent des méthodes de résolution bien particulières (méthode de Newton) qui sont extrêmement coûteuses. Il y a un vrai défi à améliorer ces méthodes en terme de rapidité d'exécution. Ce chapitre étudie l'amélioration de ces méthodes en proposant des schémas numériques nouveaux mais propose également la construction pas à pas d'un solveur rapide et efficace. Ainsi le calcul du premier point, le schéma numérique principal, l'estimation de l'erreur et la gestion des pas de temps et enfin la gestion des évènements sont successivement étudiés. Puis nous présenterons la méthode WFR appliquée au schéma d'Euler.

Sommaire

2.1 Schéma de démarrage optimisé

2.1.1 Problématique

Connaissant les conditions initiales du problème à intégrer, on commence par calculer le premier point. Il vient alors deux difficultés.

- La première est d'utiliser un schéma à un pas (car on ne connaît que la condition initiale) qui soit A-stable.
- La deuxième est sur le choix du pas de temps de départ. Un choix de pas trop petit ralentit le démarrage de la résolution du problème tandis qu'un pas de temps trop grand peut rendre le schéma instable si celui-ci n'est pas A-stable.

2.1.2 Cas des systèmes peu raides

Dans les modèles comportant de nombreuses discontinuités, c'est-à-dire les modèles pour lesquels il est nécessaire de redémarrer plusieurs fois le solveur pendant la simulation, il est important d'utiliser un schéma de démarrage qui soit rapide car il est utilisé de nombreuses fois.

Cette initialisation ne peut être effectuée qu'avec un schéma à un pas de type Runge-Kutta. Suivant la raideur du système on peut utiliser soit des schémas implicites, soit explicites.

Nous proposons un nouveau schéma de Runge-Kutta emboîté explicite permettant un démarrage rapide de la résolution pour des problèmes peu raides.

On montre dans le Chapitre 3 que pour les modèles de grande taille, le coût principal d'intégration se situe dans le calcul de la dérivée et l'inversion du système non linéaire. Il est donc important de minimiser le nombre d'appels à cette dérivée. Dans les schémas de type Runge-Kutta, les appels à la dérivée se font au niveau des étapes intermédiaires. Il est donc important de minimiser le nombre d'étapes intermédiaires. De plus, on a besoin d'une solution à deux ordres pour estimer l'erreur et ainsi choisir le bon pas de temps. On utilise pour cela des schémas emboîtés. Ces schémas ont la forme suivante :

Définition 2.1. Schéma de Runge-Kutta explicite emboîté On appelle schéma de Runge-Kutta explicite emboîté à s étages, la méthode définie par le schéma

$$\begin{aligned}
k_1 &= f(t_0, y_0),\\
k_2 &= f(t_0 + c_1 h_n, y_0 + h_n a_{21} k_1),\\
&\vdots\\
k_s &= f(t_0 + c_{s-1} h_n, y_0 + h_n \sum_{i=1}^{s-1} a_{si} k_i),\\
y_{n+1} &= y_n + h_n \sum_{i=1}^{s} b_i k_i,\\
\tilde{y}_{n+1} &= \tilde{y}_n + h_n \sum_{i=1}^{s} \tilde{b}_i k_i.
\end{aligned}$$

y_{n+1} et $\tilde{y}_{n+1}$ sont deux évaluations de la solution à t_{n+1} à deux ordres différents.

On choisit d'initialiser en utilisant un schéma Runge-Kutta d'ordre 3 et 2 emboîtés avec trois étapes intermédiaires :

$$\begin{aligned}
k_1 &= f(t_0, y_0),\\
k_2 &= f(t_0 + c_1 h_n, y_0 + h_n a k_1),\\
k_3 &= f(t_0 + c_2 h_n, y_0 + h_n b k_1 + h_n c k_1),\\
y_{n+1} &= y_n + h_n d_1 k_1 + h_n d_2 k_2 + h_n d_3 k_3,\\
\tilde{y}_{n+1} &= \tilde{y}_n + h_n \tilde{d_1} k_1 + h_n \tilde{d_2} k_2 + h_n \tilde{d_3} k_3.
\end{aligned}$$

Pour déterminer les coefficients $a, b, c, c_1, c_2, d_1, d_2, d_3, \tilde{d_1}, \tilde{d_2}, \tilde{d_3}$ on calcule l'erreur de troncature (appliquée à l'équation $y' = \lambda y$) et on annule les coefficients qui permettent d'obtenir un schéma d'ordre 3 et d'ordre 2. L'erreur de troncature de ce schéma est pour l'ordre 3 :

$$\tau_n = \bar{y}(t_n) \left((d_1 + d_2 + d_3 - 1)\lambda + (d_2 a + d_3(b + c) - \frac{1}{2})\lambda^2 h + (d_3 ca - \frac{1}{6})\lambda^3 h^2 - \frac{1}{24}\lambda^4 h^3 \right) \tag{2.2}$$

Donc ce schéma est d'ordre 3 si et seulement si

$$\begin{aligned} d_1 + d_2 + d_3 - 1 &= 0, \\ d_2 a + d_3(b+c) - \frac{1}{2} &= 0, \\ d_3 ca - \frac{1}{6} &= 0. \end{aligned}$$

Ce système est surdéterminé et on fixe des valeurs pour a, b et c. Et on a alors :

$$\begin{aligned} a &= \frac{1}{2}, \\ b &= 0, \\ c &= \frac{1}{2}, \\ d_1 &= 0, \\ d_2 &= \frac{1}{3}, \\ d_3 &= \frac{2}{3}. \end{aligned}$$

L'erreur de troncature pour le schéma d'ordre 2 est :

$$\tau_n = \bar{y}(t_n)\left((\tilde{d}_1 + \tilde{d}_2 + \tilde{d}_3 - 1)\lambda + (\tilde{d}_2 a + \tilde{d}_3(b+c) - \frac{1}{2})\lambda^2 h + (\tilde{d}_3 ca - \frac{1}{6})\lambda^3 h^2\right) \quad (2.3)$$

Donc ce schéma est d'ordre 2 si et seulement si

$$\begin{aligned} \tilde{d}_1 + \tilde{d}_2 + \tilde{d}_3 - 1 &= 0, \\ \tilde{d}_2 a + \tilde{d}_3(b+c) - \frac{1}{2} &= 0. \end{aligned}$$

Ce système est surdéterminé et on fixe des valeurs pour a, b, c et $\tilde{d}_3$. Afin que le schéma soit emboîté, on reprend les mêmes valeurs de a, b et c que pour l'ordre 3. Et on a donc le système, en choisissant $\tilde{d}_3 = 0$:

$$\begin{aligned} \tilde{d}_1 + \tilde{d}_2 + \tilde{d}_3 - 1 &= 0, \\ \tilde{d}_2 \frac{1}{2} - \frac{1}{2} &= 0. \end{aligned}$$

On a alors :

$$\begin{aligned} a &= \frac{1}{2}, \\ b &= 0, \\ c &= \frac{1}{2}, \\ \tilde{d}_1 &= 0, \\ \tilde{d}_2 &= 1, \\ \tilde{d}_3 &= 0. \end{aligned}$$

La stabilité de ce nouveau schéma est étudiée et est donnée sur la Figure 2.1 et la Figure 2.2.

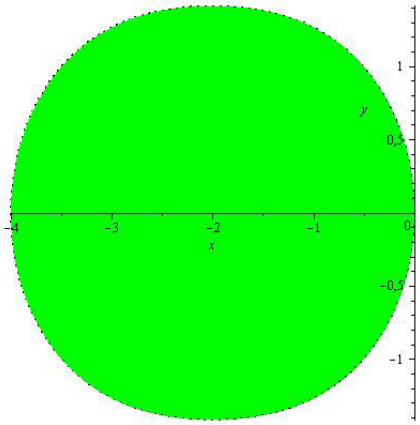


Figure 2.1 – Zone de stabilité de l'ordre 3 du nouveau schéma RK3-2

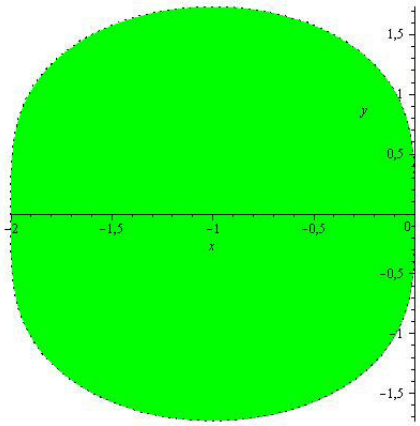


Figure 2.2 – Zone de stabilité de l'ordre 2 du nouveau schéma RK3-2

On constate que ce schéma n'est pas A-stable, et cela était prévisible du fait du caractère explicite du schéma. Dans les cas des systèmes raides, afin de ne pas avoir un pas de temps restreint à une très petite valeur on choisit un schéma de Runge-Kutta implicite.

2.1.3 Schémas DIRK et SDIRK pour les systèmes raides

Le schéma développé dans la Section 2.2 et utilisé pour tout le reste de la simulation est particulièrement efficace pour les systèmes raides et donc il est important que le schéma de démarrage soit également efficace pour les systèmes raides. En particulier, il est nécessaire que ce schéma soit A-stable afin de garantir la plus grande stabilité possible.
De manière générale, il n'est pas possible de trouver un schéma explicite A-stable. Pour établir un schéma numérique à un pas qui soit A-stable, il faut donc s'orienter vers les schémas de type Runge-Kutta implicites. Ces schémas ont la forme suivante :

$$k_i = hf(t_n + c_i h, y_n + \sum_{j=1}^{s} a_{ij} k_j), \qquad i = 1, .., s,$$

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i.$$

Le tableau de Butcher associé à ce type de schéma est le suivant :

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

On remarque qu'à chaque pas de temps on a un système algébrique à résoudre, ce qui rend ce schéma très coûteux. Le grand avantage de ce schéma est qu'il est A-stable. Un compromis intéressant entre rapidité et stabilité est de considérer le schéma à deux étages suivant :

Définition 2.2. Considérons le schéma Runge-Kutta implicite à un pas et deux étages suivant

$$\begin{aligned} k_1 &= y_n + \alpha h f(t_n, k_1) + \gamma h f(t_n, k_2), \\ k_2 &= y_n + \beta h f(t_n, k_1) + \delta h f(t_n, k_2)), \\ y_{n+1} &= y_n + h_n \sum_{i=1}^{2} b_i f(k_i). \end{aligned}$$

Le tableau de Butcher correspondant s'écrit :

$$\begin{array}{c|cc} \alpha+\gamma & \alpha & \gamma \\ \beta+\delta & \beta & \delta \\ \hline & b_1 & b_2 \end{array}$$

Une forme particulière de ces schémas Runge-Kutta implicites sont les schémas DIRK. Ce sont ces types de schémas qui vont nous intéresser.

Les schémas Diagonal Implicit Runge Kutta (DIRK)

Les schémas DIRK sont attrayants pour leur simplicité car le système à résoudre pour chaque étape intermédiaire est plus simple. En effet, l'étape intermédiaire k_i ne dépend que des étapes intermédiaires précédentes et de k_i (et pas des étapes intermédiaires suivantes comme pour les schémas IRK). On a alors un schéma à deux étages :

Définition 2.3. Considérons le schéma Runge-Kutta implicite à un pas et deux étages suivant

$$\begin{aligned} k_1 &= y_n + \alpha h f(t_n, k_1), \\ k_2 &= y_n + \beta h f(t_n, k_1) + \alpha h f(t_n, k_2)), \\ y_{n+1} &= y_n + h_n \sum_{i=1}^{2} b_i f(k_i). \end{aligned}$$

Le tableau de Butcher correspondant s'écrit :

$$\begin{array}{c|cc} \alpha & \alpha & \\ \beta+\alpha & \beta & \alpha \\ \hline & b_1 & b_2 \end{array}$$

Dans le cadre de la recherche d'un schéma de démarrage optimal, à partir du schéma (2.3) on cherche à déterminer les quatre coefficients α, β, b_1 et b_2 afin d'obtenir un schéma d'intégration le plus stable possible et d'ordre maximal. Pour cela appliquons au schéma (2.3) l'équation test de Dahlquist. On peut alors résoudre l'équation implicite en k_1.

$$k_1 = y_n + \alpha\ h\ \lambda\ k_1 \Longleftrightarrow k_1 = -\frac{y_n}{-1 + h\alpha\lambda}. \tag{2.4}$$

On remarque que k_1 ne dépend que de α.
On procède de même pour k_2 :

$$\begin{aligned} k_2 &= y_n + h\lambda(-\beta\frac{y_n}{-1 + h\alpha\lambda} + \alpha k_2) \\ \Longleftrightarrow k_2 &= -\frac{y_n(-1 + h\ \alpha\lambda - h\ \beta\lambda)}{(-1 + h\alpha\lambda)^2}. \end{aligned}$$

k_2 ne dépend que de α et β.
y_{n+1} s'exprime alors :

$$\begin{aligned} y_{n+1} &= y_n + h(b_1\lambda k_1 + b_2\lambda k_2) \\ &= \frac{y_n\left(h^2\lambda^2(\alpha^2 - b_1\alpha - b_2\alpha + b_2\beta) + h\lambda(-2\alpha + b_1 + b_2) + 1\right)}{(-1 + h\alpha\lambda)^2}. \end{aligned} \tag{2.5}$$

Si on note $\bar{y}(t_n)$ la solution exacte du problème en t_n et qu'on détermine y_{n+1} à l'aide de (2.5) on a la solution $\tilde{y}_{n+1}$ ainsi obtenue avec le schéma :

$$\tilde{y}_{n+1} = \frac{\bar{y}(t_n)\left(h^2\lambda^2(\alpha^2 - b_1\alpha - b_2\alpha + b_2\beta) + h\lambda(-2\alpha + b_1 + b_2) + 1\right)}{(-1 + h\alpha\lambda)^2}. \tag{2.6}$$

Ecrivons maintenant le développement de Taylor à l'ordre 4 de $\bar{y}(t_{n+1})$:

$$\begin{aligned} \bar{y}(t_{n+1}) &= \bar{y}(t_n) + h\frac{\mathrm{d}\bar{y}}{\mathrm{d}t} + \frac{h^2}{2}\frac{\mathrm{d}^2\bar{y}}{\mathrm{d}t^2} + \frac{h^3}{6}\frac{\mathrm{d}^3\bar{y}}{\mathrm{d}t^3} + \frac{h^4}{24}\frac{\mathrm{d}^4\bar{y}}{\mathrm{d}t^4} + O(h^5) \\ &= \bar{y}(t_n) + h\lambda\bar{y}(t_n) + \frac{h^2}{2}\lambda^2\bar{y}(t_n) + \frac{h^3}{6}\lambda^3\bar{y}(t_n) + \frac{h^4}{24}\lambda^4\bar{y}(t_n) + O(h^5). \end{aligned}$$

Etudions maintenant l'erreur locale due au schéma, c'est-à-dire la différence :

$$\begin{aligned} \frac{\bar{y}(t_{n+1}) - \tilde{y}_{n+1}}{h} &= \frac{\bar{y}(t_n)}{h}\left(1 + h\lambda + \frac{h^2}{2}\lambda^2 + \frac{h^3}{6}\lambda^3 + \frac{h^4}{24}\lambda^4 + O(h^5) - \right. \\ &\qquad \left. \frac{\left(h^2\lambda^2(\alpha^2 - b_1\alpha - b_2\alpha + b_2\beta) + h\lambda(-2\alpha + b_1 + b_2) + 1\right)}{(-1 + h\alpha\lambda)^2}\right) \\ &= \frac{\bar{y}(t_n)}{h(-1 + h\alpha\lambda)^2}\left((1 + h\lambda + \frac{h^2}{2}\lambda^2 + \frac{h^3}{6}\lambda^3 + \frac{h^4}{24}\lambda^4 + O(h^5))(-1 + h\alpha\lambda)^2 - \right. \\ &\qquad \left.\Big(h^2\lambda^2(\alpha^2 - b_1\alpha - b_2\alpha + b_2\beta) + h\lambda(-2\alpha + b_1 + b_2) + 1\Big)\right) \\ &= \frac{\bar{y}(t_n)}{h(-1 + h\alpha\lambda)^2}\left((1 + (-2\alpha + 1)\lambda h + (\alpha^2 - 2\alpha + \frac{1}{2})\lambda^2h^2 + (\alpha^2 - \alpha + \frac{1}{6})\lambda^3h^3 + O(h^4)) - \right. \\ &\qquad \left.\Big(h^2\lambda^2(\alpha^2 - b_1\alpha - b_2\alpha + b_2\beta) + h\lambda(-2\alpha + b_1 + b_2) + 1\Big)\right) \\ &= \frac{\bar{y}(t_n)}{(-1 + h\alpha\lambda)^2}\left((1 - b_1 - b_2)\lambda + ((-2 + b_1 + b_2)\alpha - b_2\beta + \frac{1}{2})\lambda^2h + (\alpha^2 - \alpha + \frac{1}{6})\lambda^3h^2 + O(h^3))\right) \end{aligned}$$

Le schéma est d'ordre 3 si et seulement si $\frac{\bar{y}(t_{n+1}) - \tilde{y}_{n+1}}{h} = O(h^3)$. Donc on doit avoir

$$\begin{cases} 1 - b_1 - b_2 = 0 \\ (-2 + b_1 + b_2)\alpha - b_2\beta + \frac{1}{2} = 0 \\ \alpha^2 - \alpha + \frac{1}{6} = 0. \end{cases}$$

Le système se réécrit

$$\begin{cases} b_1 + b_2 = 1 \\ -\alpha - b_2\beta + \frac{1}{2} = 0 \\ \alpha^2 - \alpha + \frac{1}{6} = 0. \end{cases}$$

La dernière équation donne directement deux valeurs de α possibles :

$$\alpha_{1,2} = \frac{1}{2} \pm \frac{\sqrt{3}}{6}$$

En traçant les zones d'absolue stabilité, on remarque que seule la valeur $\alpha = \alpha_1$ permet d'avoir un schéma A-stable ; on choisit donc $\alpha = \alpha_1$. En choisissant par exemple $b_1 = \frac{1}{2}$ et $b_2 = \frac{1}{2}$, on obtient alors directement la valeur de β

$$\beta = 1 - 2\alpha \tag{2.7}$$

Etudions maintenant la L-stabilité. Reprenons l'équation (2.5) et déterminons la limite de

$$\begin{aligned} \lim_{h\lambda\to\infty} \frac{y_{n+1}}{y_n} &= \lim_{h\lambda\to\infty} \frac{h^2\lambda^2(\alpha^2 - b_1\alpha - b_2\alpha + b_2\beta) + h\lambda(-2\alpha + b_1 + b_2) + 1}{(-1 + h\alpha\lambda)^2} \\ &= \frac{\alpha^2 - \alpha + \frac{1}{2}(1 - 2\alpha)}{\alpha} \\ &= \frac{\alpha^2 - 2\alpha + 1/2}{\alpha} \end{aligned}$$

Le schéma est L-stable si et seulement si $\lim_{h\lambda\to\infty} \frac{y_{n+1}}{y_n} = 0$.

On doit donc avoir $\alpha^2 - 2\alpha + \frac{1}{2} = 0$ ce qui correspond à deux valeurs $1 + \frac{\sqrt{2}}{2}$ et $1 - \frac{\sqrt{2}}{2}$. Ces deux valeurs sont différentes des valeurs de α choisies pour que le schéma soit d'ordre 3. On ne peut donc pas avoir L-stabilité et l'ordre 3.

Le choix entre les deux valeurs $1 + \frac{\sqrt{2}}{2}$ et $1 - \frac{\sqrt{2}}{2}$ est gouverné par le terme de degré 3 à savoir $\alpha^2 - \alpha + \frac{1}{6}$. On choisit la valeur de α qui donne la plus petite erreur d'ordre 3 et c'est $1 - \frac{\sqrt{2}}{2}$. En résumé, il y a donc deux possibilités de schéma de type DIRK à deux étages :

- Un schéma d'ordre 3 à deux étages A-stable dont le tableau de coefficients est le suivant :

$$\begin{array}{c|cc} \frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{2} + \frac{\sqrt{3}}{6} & \\ \frac{1}{2} - \frac{\sqrt{3}}{6} & -\frac{\sqrt{3}}{2} & \frac{1}{2} + \frac{\sqrt{3}}{6} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \tag{2.8}$$

- Un schéma d'ordre 2 à deux étages L-stable dont le tableau de coefficients est le suivant :

$$\begin{array}{c|cc} 1 - \frac{\sqrt{2}}{2} & 1 - \frac{\sqrt{2}}{2} & \\ \frac{\sqrt{2}}{2} & -1 + \sqrt{2} & 1 - \frac{\sqrt{2}}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \tag{2.9}$$

- Il n'existe donc pas de schéma SDIRK d'ordre 3 L-stable à deux étages. En effet, il manque un degré de liberté afin d'avoir la L-stabilité. Pour l'obtenir il faut considérer un schéma complètement implicite à 2 étages.

Implémentation pratique du solveur

En pratique, le démarrage s'implémente de la façon suivante :
- on choisit un pas de temps arbitraire, relativement petit, 10^{-4} par exemple,
- on donne une première évaluation du premier point avec le schéma (2.9),
- on donne une seconde évaluation du premier point avec le schéma (2.8),
- la différence entre les deux évaluations permet d'adapter le pas de temps (voir 1.1.6) et de calculer le pas de temps optimal correspondant à l'erreur maximale permise par l'utilisateur,
- on recalcule à nouveau le premier point en utilisant ce pas de temps optimal.

Le choix de recalculer à nouveau le premier point se justifie de la manière suivante :
- le pas de temps choisi arbitrairement peut être trop grand et donner une erreur supérieure à la tolérance,
- dans le cas où l'erreur est plus petite que la tolérance, on recalcule quand même le premier point afin d'éviter que le ratio entre le pas de temps optimal et le pas de temps arbitrairement choisi soit trop grand et donne lieu à des problèmes de stabilité qui sont détaillés dans la Section 2.3.3 de ce même Chapitre.

L'erreur commise lors de la différence entre les deux solutions données par les deux schémas est celle du schéma d'ordre le moins élevé, c'est-à-dire :

$$\epsilon_n = \bar{y}(t_n)(\alpha^2 - \alpha + \frac{1}{6})h^2. \tag{2.10}$$

Cette erreur correspond à celle du schéma d'ordre 2, donc $\alpha = 1 - \frac{\sqrt{2}}{2}$, et on a :

$$\epsilon_n = -0.0404\bar{y}(t_n)\lambda^2 h^2. \tag{2.11}$$

On compare par rapport à un schéma de Runge-Kutta 3-2 emboîté pour lequel l'erreur est :

$$\tilde{\epsilon}_n = -\frac{1}{6}\lambda^2 h^2, \tag{2.12}$$

ce qui donne :

$$\tilde{\epsilon}_n = -0.1667\lambda^2 h^2. \tag{2.13}$$

Etude de la stabilité

En reprenant le schéma d'ordre 3 A-stable et l'équation (2.6) et en posant $z = \lambda h$, on a

$$\frac{y_{n+1}}{y_n} = \frac{z^2(\alpha^2 - \alpha + \frac{1}{2}\beta) + z(-2\alpha + 1) + 1}{(-1 + z\alpha)^2}, \tag{2.14}$$

avec $\alpha = \frac{1}{2} + \frac{\sqrt{3}}{6}$ et $\beta = -\frac{\sqrt{3}}{2}$.

La zone de stabilité est représentée sur la Figure 2.3, à l'extérieur du contour :

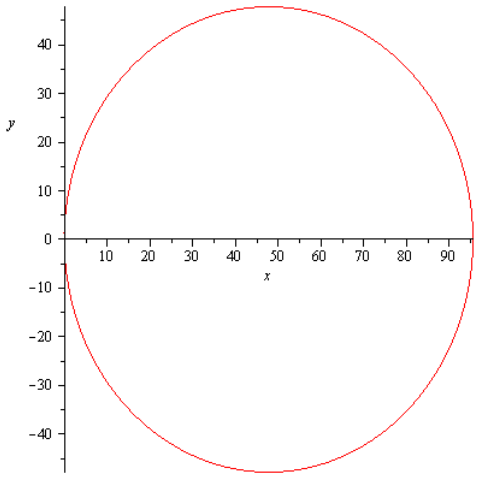


FIGURE 2.3 – Zone de stabilité du DIRK3

On remarque que le schéma est bien A-stable.

Il en est de même pour le schéma DIRK d'ordre 2 L-stable, pour lequel on représente la zone de stabilité à l'extérieur du contour de la Figure 2.4.

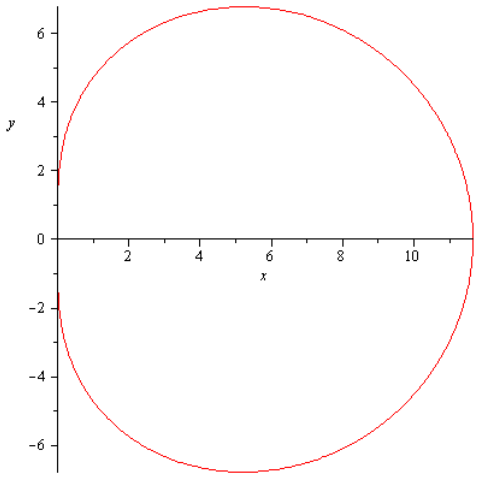


FIGURE 2.4 – Zone de stabilité du DIRK2

Ce schéma est bien A-stable également.

2.1.4 Calcul du pas de temps optimal

Le calcul du pas de temps optimal reprend la méthode décrite dans (1.1.6) à partir des schémas SDIRK d'ordre 2 et 3.

2.1.5 Comparaison avec schéma d'Euler explicite

On compare maintenant le schéma de démarrage dit DIRK 3-2 (deux évaluations d'ordre 3 et 2 à l'aide de schémas DIRK et adaptation du pas de temps) avec un schéma

d'Euler avec un pas de temps très petit. En particulier on s'intéresse à l'évolution du pas de temps pour les deux méthodes sur la Figure 2.5 :

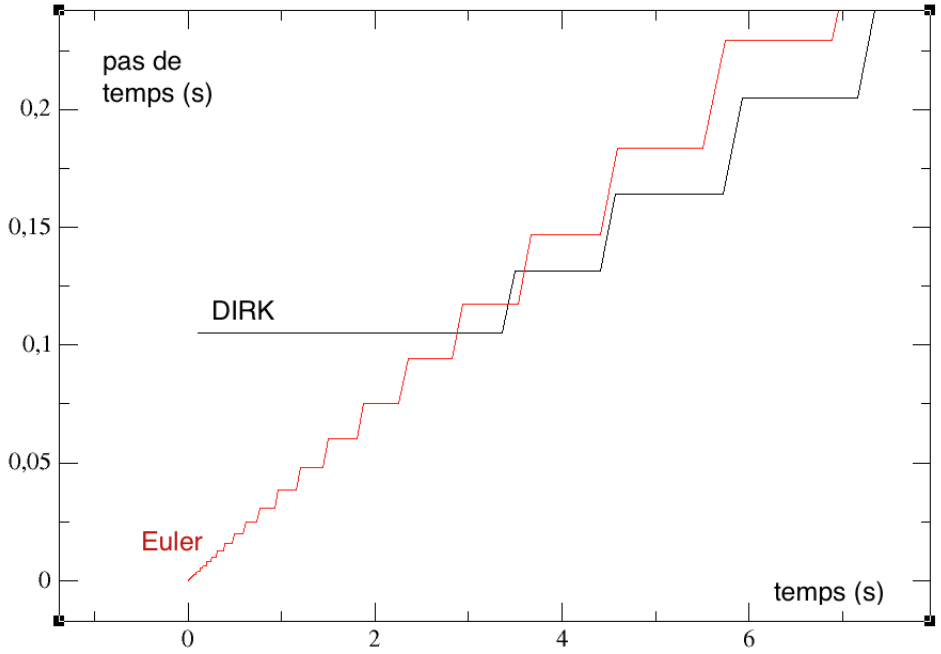


Figure 2.5 – Evolution du pas de temps au début de la simulation avec Euler (en rouge) et DIRK 3-2 (en noir).

On remarque que le schéma DIRK démarre avec un pas de temps beaucoup plus grand que le schéma d'Euler (tout en respectant l'erreur maximale permise) et est donc plus rapide car il a besoin de moins de points. L'adaptation du pas de temps du schéma DIRK montre ici tout son avantage par rapport au démarrage en pas fixe utilisant le schéma d'Euler avec un pas de temps très petit.

Le schéma d'intégration utilisé en dehors du démarrage doit maintenant être choisi ; ce schéma est appelé schéma courant.

2.2 Nouveau schéma principal d'intégration LIBDF

Dans cette section, on présente le schéma principal d'intégration numérique utilisé que l'on a décidé de nommer LIBDF. Avant de détailler ce nouveau schéma, il est important de décrire l'importance des systèmes non linéaires, raides et de grande taille dans les systèmes physiques industriels. En effet, beaucoup de problèmes en physique, en ingénierie, en chimie, en biologie contiennent des EDOs raides. De plus, certains de ces systèmes physiques, en particulier en mécanique des fluides, en milieux poreux, dans les océans, sont issus de discrétisations spatiales d'équations aux dérivées partielles non linéaires. On a vu dans la section 1.1.4 du Chapitre 1 que la raideur de tels systèmes était caractérisée par le ratio entre la plus grande et la plus petite valeur propre de la matrice Jacobienne $J = \frac{\partial f}{\partial \bar{y}}$. On a également vu, toujours dans la même section, que les schémas explicites ne permettaient pas l'utilisation de pas de temps élevés pour de tels systèmes et c'est la raison pour laquelle les schémas implicites étaient utilisés. Parmi ces schémas implicites,

les schémas BDF sont particulièrement intéressants de par leur bonne absolue stabilité (voir section 1.1.4 du Chapitre 1). Cependant ces schémas nécessitent la résolution de la partie implicite. On a vu dans la section 1.1.5 du Chapitre 1 que cette résolution était soit très coûteuse, soit conduisait à un schéma moins stable que celui de départ. L'idée du nouveau schéma développé dans cette partie est de conserver les propriétés exactes de stabilité du schéma BDF, de limiter l'erreur globale en utilisant les points d'équilibre du système et de réduire de manière importante les coûts de calcul en linéarisant la partie implicite du schéma.

2.2.1 Considérations

On introduit tout d'abord le contexte et les conditions dans lesquelles on se place.

Comme $f \in C^p(\mathbb{R}^d, \mathbb{R}^d)$, alors grâce au théorème de Cauchy-Lipschitz (cf section 1.1.1 du Chapitre 1, et pour plus de détails voir [CL55]), il existe $T_0 > 0$ tel que l'IVP (2.1) admette sur $[0, T_0]$ une unique solution $\bar{y} \in C^{p+1}([0, T_0], \mathbb{R}^d)$.

Ensuite, on choisit T tel que $0 < T < T^*$, où T^* est le temps maximal d'existence de la solution y. On suppose de plus qu'il existe une constante $C_0 > 0$ qui ne dépend pas de T telle que

$$\max_{0 \leq k \leq p+1} \sup_{t \in [0,T]} |\bar{y}^{(k)}(t)| \leq C_0. \tag{2.15}$$

Soit une fonction $g \in C^k([0, T])$, on note $M_k(g)$ la valeur $M_k(g) := \sup_{t \in [0,T]} |g^{(k)}(t)|$.

On introduit $\mathfrak{E}_f$ l'ensemble des points d'équilibre de f défini comme $\mathfrak{E}_f := \{x \in \mathbb{R}^d;\ f(x) = 0\}$; cet ensemble peut éventuellement être vide. On introduit alors $\mathfrak{S}_f$ l'ensemble des points d'équilibre de f défini par $\mathfrak{S}_f := \{x \in \mathfrak{E}_f;\ f'(x) < 0\}$.

2.2.2 Le schéma Linearized Interpolated Backward Differentiation Formulae

Pour $N \in \mathbb{N}^*$ et une suite croissante $\{t_i\}_{0 \leq i \leq N}$ subdivision de l'intervalle $[0, T]$ tel que $t_0 = 0$, $t_N = T$ et $h_i = t_{i+1} - t_i$, $0 \leq i \leq N-1$ le numéro du pas de temps, les méthodes BDF d'ordre p s'expriment sous la forme (voir section 1.1.4 du Chapitre 1 et [Ske86] et [JSD80]) : pour tout $p - 1 \leq n \leq N - 1$,

$$y_{n+1} = \sum_{i=0}^{p-1} \alpha_{i,n} y_{n-i} + \beta_n h_n f(y_{n+1}), \tag{2.16}$$

avec les coefficients variables $\alpha_{i,n}$ et β_n fonctions du ratio de pas de temps $\omega_i = h_i / h_{i-1}$ pour $i = n, ..., n + 2 - p$ (voir [CGG90] ainsi que la Section 2.3.3 pour le calcul des expressions des coefficients).

Pour une suite $\boldsymbol{u} = \{u_i\}_{i \in [\![0,N]\!]}$ et pour $p - 1 \leq n \leq N - 1$, on note par $P_{p,n}(t, \boldsymbol{u})$ l'interpolation polynomiale de Lagrange (voir Définition 6.1 de [SM03]) de degré $p - 1$ pour les points $\{(t_{n-i}, u_{n-i}) : i = 0, 1, .., p - 1\}$.

On note par $\mathcal{P}_{p,n}(\boldsymbol{u})$ la valeur $P_{p,n}(t_{n+1}, \boldsymbol{u})$.

Soit une fonction $g \in C^p([0, T])$ et soit $\boldsymbol{u} = \{g(t_i)\}_{i \in [\![0,N]\!]}$, alors grâce au Théorème 6.2 de [SM03], on a :

$$|\mathcal{P}_{p,n}(\boldsymbol{u}) - g(t_{n+1})| \leq \frac{M_p(g)}{p!} \prod_{i=0}^{p-1} (t_{n+1} - t_{n-i}), \tag{2.17}$$

et avec (2.17), pour une suite $\boldsymbol{u} = \{u_i\}_{i \in [\![0,N]\!]}$ et pour $p - 1 \leq n \leq N - 1$, on dit que $\mathcal{P}_{p,n}(\boldsymbol{u})$ est une approximation de u_{n+1} d'ordre p.

Expression du schéma LIBDF

L'idée de la construction de notre schéma p-LIBDF repose sur une approximation d'ordre p du terme implicite $f(y_{n+1})$:

$$f(y_{n+1}) \approx f(\mathcal{P}_{p,n}(\boldsymbol{y})) + \mathcal{A}_n(\boldsymbol{y})(y_{n+1} - \mathcal{P}_{p,n}(\boldsymbol{y})), \tag{2.18}$$

avec $\boldsymbol{y} = \{y_i\}_{i\in[\![0,N]\!]}$ et $\mathcal{A}_n(\boldsymbol{y})$ une matrice de taille d telle que $\mathcal{A}_n(\boldsymbol{y}) \in \{f'(c); c \in \mathfrak{E}_f\} \cup \{f'(\mathcal{P}_{p,n}(\boldsymbol{y}))\}$. De même que (2.16), on obtient notre nouveau schéma :

$$y_{n+1} = \sum_{i=0}^{p-1} \alpha_{i,n} y_{n-i} + \beta_n h_n (f(\mathcal{P}_{p,n}(\boldsymbol{y})) + \mathcal{A}_n(\boldsymbol{y})(y_{n+1} - \mathcal{P}_{p,n}(\boldsymbol{y}))). \tag{2.19}$$

Un choix possible pour $\mathcal{A}_n(\boldsymbol{y})$ est le suivant : pour tout $n \in \mathbb{N}^*$, $n \geq p-1$,
- si $\mathfrak{E}_f \neq \emptyset$ et qu'il existe $c \in \mathfrak{E}_f$ tel que
  $|y_n - c| \leq \max\{y_{n-1} - c|, ..., |y_{n-p} - c|\}$ alors $\mathcal{A}_n(\boldsymbol{y}) = f'(c)$,
- sinon $\mathcal{A}_n(\boldsymbol{y}) = f'(\mathcal{P}_{p,n}(\boldsymbol{y}))$. Si $p = 1$, $\mathcal{A}_0(\boldsymbol{y}) = f'(y_0)$.

Ce choix de valeur de $\mathcal{A}_n(\boldsymbol{y})$ est motivé par le fait que si $\mathcal{A}_n(\boldsymbol{y}) = f'(\mathcal{P}_{p,n}(\boldsymbol{y}))$, alors notre schéma correspond à la première itération de l'algorithme de Newton appliqué pour résoudre l'implicitation du schéma p-BDF (2.16) avec une valeur initiale égale à $\mathcal{P}_{p,n}(\boldsymbol{y})$. Cependant si la suite $\{y_n\}$ converge, alors grâce à (2.19), elle converge vers un certain $\ell \in \mathfrak{E}_f$ et alors pour un certain $N \in \mathbb{N}$, pour tout $n \geq N$ au lieu d'utiliser $\mathcal{A}_n(\boldsymbol{y}) = f'(\mathcal{P}_{p,n}(\boldsymbol{y}))$, on économise en temps de calcul CPU en utilisant directement $\mathcal{A}_n(\boldsymbol{y}) = f'(\ell)$.
Comme le terme implicite est maintenant linéarisé, le schéma (2.19) peut être réécrit explicitement comme :

Définition 2.4. Le schéma p-LIBDF est le suivant : $\forall p \in \mathbb{N}^*$, pour tout $n \geq p-1$,

$$y_{n+1} = (I - \beta_n h_n \mathcal{A}_n(\boldsymbol{y}))^{-1} \left(\sum_{i=0}^{p-1} \alpha_{i,n} y_{n-i} + \beta_n h_n (f(\mathcal{P}_{p,n}(\boldsymbol{y})) - \mathcal{A}_n(\boldsymbol{y}) \mathcal{P}_{p,n}(\boldsymbol{y})) \right). \tag{2.20}$$

Dans un souci de simplicité, on considère un espace de dimension $d = 1$ et notre analyse est faite avec un pas de temps fixe. Dans le cas où le pas de temps est fixe, on a pour tout $0 \leq i \leq N$, $t_i = ih$, avec $h = \frac{T}{N}$ l'unique pas de temps. On remarque alors que les coefficients variables $\alpha_{i,n}$ et β_n ne dépendent plus de l'indice n, et dans ce cas la définition 2.20 devient :

Définition 2.5. Le schéma p-LIBDF est le suivant : $\forall p \in \mathbb{N}^*$,

$$y_{n+1} = (1 - \beta h \mathcal{A}_n(\boldsymbol{y}))^{-1} \left(\sum_{i=0}^{p-1} \alpha_i y_{n-i} + \beta h (f(\mathcal{P}_{p,n}(\boldsymbol{y})) - \mathcal{A}_n(\boldsymbol{y}) \mathcal{P}_{p,n}(\boldsymbol{y})) \right), \tag{2.21}$$

avec β et $\{\alpha_i\}_{i\in[\![0,p-1]\!]}$ les coefficients définissant le schéma p-BDF usuel (2.16) dans le cas des pas de temps constants. A partir du Lemme 2.3 de [Ise96], ces coefficients sont donnés par $\beta = \left(\sum_{\ell=1}^{p} \frac{1}{\ell} \right)^{-1}$ et pour tout $0 \leq i \leq p-1$,

$$\alpha_i = \beta (-1)^i \sum_{\ell=i+1}^{p} \frac{C_\ell^{i+1}}{\ell}.$$

De plus, pour toute suite $\boldsymbol{u} = \{u_i\}_{i\in[\![0,N]\!]}$ et pour tout $p-1 \leq n \leq N-1$, on remarque que si on applique la formule donnée par Lagrange dans [Lag77],

$$\mathcal{P}_{p,n}(\boldsymbol{u}) = \sum_{k=0}^{p-1}(-1)^k C_p^{k+1} u_{n-k}. \tag{2.22}$$

De plus, à partir de (2.17), on déduit que pour une fonction $g \in C^p([0,T])$ et pour $\boldsymbol{u} = \{g(ih)\}_{i\in[\![0,N]\!]}$, on a

$$|\mathcal{P}_{p,n}(\boldsymbol{u}) - g((n+1)h)| \leq M_p(g)h^p. \tag{2.23}$$

Dans ce qui suit, on suppose que $0 < h \leq 1$.

2.2.3 Stabilité du schéma p-LIBDF

Dans ce paragraphe on donne deux propriétés de stabilité pour le schéma LIBDF.

Zéro-Stabilité

On donne ici la définition générale de la zéro-stabilité d'une méthode numérique $F : \mathbb{R}^{m+1} \longmapsto \mathbb{R}^{\ell}$, $\ell \leq m$ en généralisant la formulation de la section 1.1.3 du Chapitre 1 et de Sueli [SM03]. La solution $\boldsymbol{y} = \{y_i\}_{i\in[\![0,m]\!]}$ de F vérifie l'équation $F(\boldsymbol{y}, h) = 0$. On considère maintenant $\boldsymbol{u}$ une solution de F calculée avec une erreur d'arrondi $\xi = \{\xi_j\}_{j\in[\![0,\ell]\!]} : F(\boldsymbol{y}, h) = \xi : F(\boldsymbol{y}, h) = \xi$, avec ξ suffisamment petit. Pour $m \in \mathbb{N}^*$, on note $\|\cdot\|_m$ la norme euclidienne dans $\mathbb{R}^m$.

Définition 2.6. On considère l'IVP suivante :

$$y' = 0, \qquad y(0) = y_0. \tag{2.24}$$

$\boldsymbol{y}$ et $\boldsymbol{u}$ sont ses solutions numériques définies dans le paragraphe précédent ; la méthode numérique est appelée zéro-stable si :

$$\|\boldsymbol{y} - \boldsymbol{u}\|_m \leq C\|\xi\|_{\ell}, \tag{2.25}$$

où C dépend de la longueur T de l'intervalle de simulation mais est indépendant de h.

Considérant cette définition, on a la proposition suivante :

Proposition 2.7. *Le schéma p-LIBDF est zéro-stable pour* $1 \leq p \leq 6$.

Démonstration. La preuve découle immédiatement du fait que pour $f = 0$, le schéma p-LIBDF correspond exactement au schéma p-BDF et ce dernier est stable pour $1 \leq p \leq 6$ (voir [HW83] et [Gri83]). □

Stabilité absolue

On rappelle ici la définition de la stabilité absolue d'après Dahlquist dans [Dah63] (voir aussi section 1.1.4 du Chapitre 1).

Définition 2.8. Une méthode numérique est absolument stable pour un λh donné et fixé, si toutes ses solutions tendent vers zéro quand $n \to \infty$, pour une équation différentielle de la forme

$$y' = \lambda y, \tag{2.26}$$

avec λ un complexe constant avec une partie réelle négative.

Considérant cette définition, on a la proposition suivante :

Proposition 2.9. *La région d'absolue stabilité du schéma p-LIBDF est la même que le schéma p-BDF.*

Démonstration. Les notions de régions d'absolue stabilité sont détaillées dans la section 1.1.4 du Chapitre 1 et la preuve de cette proposition découle immédiatement du fait que pour $f(y) = \lambda y$, le schéma p-LIBDF correspond exactement au schéma p-BDF (voir [Gea71], [SM03] p. 349 and [HN93] p. 246). □

Pour la suite, on suppose que $1 \leq p \leq 6$.
Ce paragraphe introduit le Lemme 2.11, qui est utilisé pour la démonstration du Théorème 2.20. On a vu dans la section 1.1.4 du Chapitre 1 que la région de stabilité absolue est l'ensemble des points λh dans le plan complexe pour lesquels la méthode est absolument stable. Alors, indifféremment, les schémas p-LIBDF et p-BDF sont absolument stables pour une valeur donnée de λh si chaque racine $z_r = z_r(\lambda h)$ du polynôme de stabilité associé $\pi(z; (\lambda h)) := \rho(z) - (\lambda h)\sigma(z)$ avec $\sigma(z) = \beta z^p$, satisfait $|z_r(\lambda h)| < 1$, avec $\rho(z) = z^p - \sum_{i=0}^{p-1} \alpha_i z^{p-1-i}$ (voir section 12.11 dans [SM03])). Soit $\mu > 0$, le polynôme de stabilité $\pi(\cdot, -\frac{\mu}{\beta})$ est vu comme le polynôme caractéristique de la matrice compagnon [1] $A_p(\mu)$ de taille p définie par (voir théorème C.3.5 de [KA01]) :

$$A_p(\mu) = \begin{pmatrix} \frac{\alpha_0}{1+\mu} & \frac{\alpha_1}{1+\mu} & \dots & \frac{\alpha_{p-2}}{1+\mu} & \frac{\alpha_{p-1}}{1+\mu} \\ 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}. \tag{2.28}$$

Ainsi, les racines $z_r(-\frac{\mu}{\beta})$ du polynôme de stabilité $\pi(\cdot; -\frac{\mu}{\beta})$ associé à la méthode p-BDF sont les valeurs propres de la matrice $A_p(\mu)$ et il est connu (voir [BM08] and [Mar66]) que ces racines satisfont $|z_r(-\frac{\mu}{\beta})| < 1$ à condition que $\mu \in \mathbb{R}$ et $\mu > 0$.

Pour $\mu = 0$, on sait que $A_p(\mu)$ est diagonalisable ; on introduit alors l'ensemble non-vide $\mathfrak{D}_p = \{\mu \in \mathbb{R}^+; A_p(\mu) \text{ est diagonalisable }\}$. En effet, étant donné que la fonction $\mathcal{F}_p : \mu \longmapsto A_p(\mu)$ est continue sur $\mathbb{R}^+$, on en déduit que la fonction $\mu \longmapsto \pi(\cdot; -\frac{\mu}{\beta})$ est continue également. On sait que pour $\mu = 0$, grâce au Corollaire C.3.6 de [KA01] (voir également la sous-section 10.3 de [Abe07]), que les valeurs propres de $A_p(\mu)$ (qui sont également les racines du polynôme $\pi(\cdot, -\frac{\mu}{\beta})$) sont toutes disctinctes. Et donc il existe $\mu_0 > 0$ tel que quel que soit $\mu \in [0, \mu_0]$ $\pi(\cdot; -\frac{\mu}{\beta})$ a toutes ses racines distinctes et que donc $A_p(\mu)$ est diagonalisable.

Quel que soit $\mu \in \mathfrak{D}_p$, $\mu > 0$, on a $A_p(\mu) = P_p(\mu) D_p(\mu) P_p(\mu)^{-1}$, avec $P_p(\mu)$ une matrice inversible de taille p où chacune de ses colonnes est un vecteur propre de $A_p(\mu)$ et $D_p(\mu)$ est la matrice diagonale de taille p composée des valeurs propres de $A_p(\mu)$. Ensuite, on définit la norme suivante sur $\mathbb{R}^p$: pour tout $x \in \mathbb{R}^p$,

$$\|x\|_\mu := \|P_p(\mu)^{-1} x\|_\infty. \tag{2.29}$$

1. On rappelle que la matrice compagnon $C(p)$ du polynôme unitaire $p(X) = c_0 + c_1 X + \dots c_{n-1} X^{n-1} + X^n$ est

$$C(p) = \begin{pmatrix} -c_{n-1} & -c_{n-2} & \dots & -c_1 & -c_0 \\ 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}. \tag{2.27}$$

On définit également la norme matricielle induite : pour tout B, matrice de taille p,

$$|||B|||_\mu := \sup_{x\in\mathbb{R}^p, x\neq 0} \frac{\|Bx\|_\mu}{\|x\|_\mu} = \|P_p(\mu)^{-1} B P_p(\mu)\|_\infty. \tag{2.30}$$

On remarque que $|||A_p(\mu)|||_\mu = |||D_p(\mu)|||_\infty = \rho_v(A_p(\mu)) < 1$ pour $\mu > 0$, la plus grande valeur propre en module de $A_p(\mu)$. Pour la preuve de notre Théorème 2.20, on a besoin du Lemme 2.11. Le Lemme 2.11 est obtenu grâce au Lemme suivant :

Lemme 2.10. *Pour tout $\mu_0 > 0$, il existe $\theta_p > 0$ dépendant seulement de p et de μ_0, tel que pour tout $\mu \in \mathfrak{D}_p$, $\mu \geq \mu_0$, on a*

$$\rho_v(A_p(\mu)) \leq \frac{1}{(1+\theta_p\mu)^{\frac{1}{p}}}.$$

Démonstration. Soit $\mu \geq \mu_0$ et $z \in \mathbb{C}$ une valeur propre de $A_p(\mu)$; ainsi z satisfait la condition suivante :

$$(1+\mu)z^p - \sum_{i=0}^{p-1} \alpha_i z^{p-1-i} = 0,$$

ce qui implique

$$(1+\mu)z^p = \sum_{i=0}^{p-1} \alpha_i z^{p-1-i}.$$

Comme $|z| \leq \rho_v(A_p(\mu)) \leq 1$, on en déduit

$$(1+\mu)|z|^p \leq \sum_{i=0}^{p-1} |\alpha_i||z|^{p-1-i} \leq \sum_{i=0}^{p-1} |\alpha_i|. \tag{2.31}$$

Alors, on obtient $|z| \leq \dfrac{\gamma_p}{1+\mu}$ avec $\gamma_p = \displaystyle\sum_{i=0}^{p-1} |\alpha_i| \geq 1$ comme $\displaystyle\sum_{i=0}^{p-1} \alpha_i = 1$. Comme $\dfrac{\gamma_p}{1+\mu} = \dfrac{1}{1+\frac{1}{\gamma_p}(\mu-(\gamma_p-1))}$, alors pour tout $\mu \geq 2(\gamma_p-1)$, on a $\mu \geq \frac{\mu}{2} + (\gamma_p - 1)$ et ainsi on déduit que si $\mu \geq 2(\gamma_p-1)$ alors $\dfrac{\gamma_p}{1+\mu} \leq \dfrac{1}{1+\frac{\mu}{2\gamma_p}}$. On pose $\mu_p = \max(2(\gamma_p-1), \mu_0)$.

Alors, en utilisant (2.31), on déduit que pour tout $\mu \geq \mu_p$,

$$|z| \leq \frac{1}{\left(1+\frac{\mu}{2\gamma_p}\right)^{\frac{1}{p}}}. \tag{2.32}$$

La fonction $g : \mu \longmapsto \rho_v(A_p(\mu))$ est continue et atteint alors son maximum sur $[\mu_0, \mu_p]$ que l'on note $\rho_{p,\max} < 1$. En posant $\alpha_p = \dfrac{\frac{1}{\rho_{p,\max}^p} - 1}{\mu_p}$, on a $\dfrac{1}{(1+\alpha_p\mu_p)^{\frac{1}{p}}} = \rho_{p,\max}$. Ainsi, on en déduit que pour tout $\mu \in [\mu_0, \mu_p]$,

$$|z| \leq \rho_v(A_p(\mu)) \leq \rho_{p,\max} = \frac{1}{(1+\alpha_p\mu_p)^{\frac{1}{p}}} \leq \frac{1}{(1+\alpha_p\mu)^{\frac{1}{p}}}. \tag{2.33}$$

En prenant $\nu_p = \min(\alpha_p, \frac{1}{2\gamma_p})$ et en utilisant (2.32),(2.33), on déduit que pour tout $\mu \geq \mu_0$,

$$|z| \leq \frac{1}{(1+\nu_p\mu)^{\frac{1}{p}}},$$

ce qui termine la preuve. □

On utilise le Lemme 2.10 pour obtenir le Lemme suivant :

Lemme 2.11. *Il existe $\alpha_p > 0$ dépendant seulement de p tel que pour tout $\mu \in \mathfrak{D}_p$, $\mu > 0$, on a,*

$$\rho_v(A_p(\mu)) \leq \frac{1}{(1+\alpha_p\mu)^{\frac{1}{p}}}.$$

Démonstration. Grâce au Théorème 2.1 dans [Ise96], on en déduit que

$$\frac{\rho(z)}{\sigma(z)\log(z)} = 1 + O(|z-1|^p), \quad z \to 1, \tag{2.34}$$

avec $\sigma(z) = \beta z^p$. L'équation (2.34) implique

$$\frac{\rho(z)}{\sigma(z)\log(z)} \to 1 \quad \text{quand } z \to 1. \tag{2.35}$$

Comme $\mu \in \mathfrak{D}_p$, la matrice compagnon $A_p(\mu)$ (2.28) est alors diagonalisable. Grâce au Corollaire C.3.6 de [KA01] (voir également la sous-section 10.3 de [Abe07]), on en déduit que les valeurs propres de $A_p(\mu)$ (qui sont également les racines du polynôme $\pi(\cdot, -\frac{\mu}{\beta})$) sont toutes disctinctes. Alors, soit $z_0(\mu) \in \mathbb{C}$ l'unique valeur propre de $A_p(\mu)$ telle que $\rho_v(A_p(\mu)) = |z_0(\mu)|$; on remarque également que $z_0(\mu)$ est une racine de $\pi(\cdot, -\frac{\mu}{\beta})$.

Pour $\mu = 0$, on en déduit :
- 1 est valeur propre de $A_p(\mu)$,
- 1 est racine simple de $\pi(\cdot, -\frac{\mu}{\beta})$, grâce à la zéro-stabilité du schéma p-BDF pour $1 \leq p \leq 6$ (voir [HW83] et théorème 12.4 de [SM03]),
- 1 est la seule valeur propre de $A_p(\mu)$ en module égal à 1, grâce à la stabilité forte du schéma p-BDF pour $1 \leq p \leq 6$ (voir Théorème 6 dans [R.79] utilisé avec $l = 1$ car la méthode de Brown (k,1) correspond à la méthode p-BDF). Ainsi on obtient également $z_0(\mu) = 1$ pour $\mu = 0$.

Alors, comme la fonction polynomiale $\mu \longmapsto \pi(\cdot, -\frac{\mu}{\beta})$ est continue à coefficients réels et que les racines de cette dernière sont toutes distinctes, on en déduit que $z_0(\mu) \to 1$ quand $\mu \to 0^+$ et qu'il existe $\mu_1 > 0$ tel que pour tout $\mu \in [0, \mu_1]$, $z_0(\mu)$ est un nombre réel.

Comme $z_0(\mu)$ est une valeur propre de $A_p(\mu)$, alors $0 = \pi(z_0(\mu), -\frac{\mu}{\beta}) = \rho(z_0(\mu)) + \frac{\mu}{\beta}\sigma(z_0(\mu))$, et on en déduit que $\dfrac{\rho(z_0(\mu))}{\sigma(z_0(\mu))} = -\dfrac{\mu}{\beta}$. Ainsi, grâce à (2.35) combiné avec le fait que $z_0(\mu) \to 1$ quand $\mu \to 0^+$, il vient

$$-\frac{\mu}{\beta \log z_0(\mu)} \to 1 \text{ quand } \mu \to 0^+.$$

On en déduit qu'il existe $0 < \mu_2 \leq \mu_1$ tel que pour tout $0 < \mu \leq \mu_2$, on a $z_0(\mu)$ réel, $0 < z_0(\mu) \leq 1$ et

$$-\frac{\mu}{\beta \log z_0(\mu)} \leq \frac{3}{2},$$

ce qui implique, comme $0 < z_0(\mu) \leq 1$,

$$0 \leq z_0(\mu) \leq e^{-\frac{2\mu}{3\beta}}. \tag{2.36}$$

Comme pour tout $x \geq 0$, $e^x \geq 1 + x$, alors pour $p \geq 1$, $e^{px} \geq 1 + x$ ce qui implique $e^{-x} \leq \frac{1}{(1+x)^{\frac{1}{p}}}$. Ainsi de (2.36), on a pour tout $0 < \mu \leq \mu_2$,

$$0 \leq z_0(\mu) \leq \frac{1}{(1 + \frac{2\mu}{3\beta})^{\frac{1}{p}}}, \tag{2.37}$$

ce qui signifie que pour tout $0 < \mu \leq \mu_2$,

$$\rho_v(A_p(\mu)) \leq \frac{1}{(1 + \frac{2\mu}{3\beta})^{\frac{1}{p}}}. \tag{2.38}$$

Et grâce au Lemme 2.10 utilisé avec $\mu_0 = \mu_2$, on conclut la preuve. □

2.2.4 Erreur de consistance et ordre du schéma p-LIBDF

On généralise la définition de la consistance et de l'ordre (voir section 1.1.3 du Chapitre 1 ou Définitions 2.5 and 2.6 dans [M$\ddot{\text{7}}$4]) :

Définition 2.12. On suppose que le schéma numérique s'écrit comme $y_{n+1} = \Psi(t_{n+1}, y_{n+1-p}, y_{n-p}, \ldots, y_n, h)$ pour tout $p-1 \leq n \leq N-1$. Pour $\bar{y} \in C^{p+1}([0,T])$ solution de (2.1), on définit l'erreur de troncature de la méthode : $\tau_n = \frac{\bar{y}(t_{n+1}) - \Psi(t_{n+1}, \bar{y}(t_{n+1-p}), \bar{y}(t_{n-p}), \ldots, \bar{y}(t_n), h)}{h}$. Alors la méthode est dite consistante si $\tau = \sup_{p-1 \leq n \leq N-1} |\tau_n|$ est tel que

$$\lim_{h \to 0} \tau = 0. \tag{2.39}$$

La méthode est d'ordre p si

$$\tau = O(h^p). \tag{2.40}$$

Considérant cette définition plus générale de la consistance et de l'ordre, on a alors la proposition 2.13. Pour démontrer cette proposition, on a besoin d'utiliser une constante de Lipschitz de f. Comme $f \in C^p(\mathbb{R})$, si on prend comme constante de Lispchitz la valeur $\sup_{z \in \mathbb{R}} |f'(z)|$, cette valeur peut être égale à l'infini. Afin d'éviter cette situation, grâce à (2.15), on introduit l'intervalle J_p défini par $J_p := [m_0(\bar{y}) - M_p(\bar{y})h^p, m_1(\bar{y}) + M_p(\bar{y})h^p]$, avec $m_0(\bar{y}) = \inf_{t \in [0,T]} \bar{y}(t)$ et $m_1(\bar{y}) = \sup_{t \in [0,T]} \bar{y}(t)$. Ensuite on introduit L_p la constante de Lispchitz de f sur J_p définie par $L_p = \sup_{z \in J_p} |f'(z)|$. Cette constante L_p est alors suffisante pour obtenir la preuve de notre Proposition 2.13.
Il est important de noter que, grâce à (2.15), il existe un intervalle J qui ne dépend pas de T tel que $J_p \subset J$, et ainsi L_p admet une borne supérieure qui ne dépend pas de T, par exemple $L = \sup_{z \in J} |f'(z)|$.

Procédons maintenant à la preuve de la Proposition 2.13.

Proposition 2.13. *Le schéma p-LIBDF est d'ordre p et son erreur de troncature locale τ_n satisfait pour tout $p-1 \leq n \leq N-1$*

$$|\tau_n| \leq \left(C_p M_{p+1}(\bar{y}) + M_p(\bar{y}) \left(2L_p + \sup_{c \in \mathfrak{E}_f} |f'(c)| \right) \right) h^p,$$

avec $C_p > 0$ une constante dépendant uniquement de p.

Démonstration. Le schéma p-LIBDF est :

$$y_{n+1} = \sum_{i=0}^{p-1} \alpha_i y_{n-i} + \beta h (f(\mathcal{P}_{p,n}(\boldsymbol{y})) + \mathcal{A}_n(\boldsymbol{y})(y_{n+1} - \mathcal{P}_{p,n}(\boldsymbol{y}))).$$

Afin d'étudier la consistance et l'ordre du schéma p-LIBDF, on considère l'erreur de troncature :

$$\tau_n = \frac{1}{h} \left(\bar{y}(t_{n+1}) - \left(\sum_{i=0}^{p-1} \alpha_i \bar{y}(t_{n-i}) + \beta h (f(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) + \mathcal{A}_n(\bar{\boldsymbol{y}})(\bar{y}(t_{n+1}) - \mathcal{P}_{p,n}(\bar{\boldsymbol{y}}))) \right) \right),$$

avec $\bar{\boldsymbol{y}}$ la suite définie par $\bar{\boldsymbol{y}} = \{\bar{y}(t_i)\}_{i \in [\![0N]\!]}$. L'erreur de troncature τ_n est réécrite comme :

$$\tau_n = \tilde{\tau}_n + \beta (f(\bar{y}(t_{n+1})) - f(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) - \mathcal{A}_n(\bar{\boldsymbol{y}})(\bar{y}(t_{n+1}) - \mathcal{P}_{p,n}(\bar{\boldsymbol{y}}))), \tag{2.41}$$

avec $\tilde{\tau}_n$ l'erreur de troncature du schéma p-BDF donnée par

$$\tilde{\tau}_n = \frac{1}{h} \left(\bar{y}(t_{n+1}) - \left(\sum_{i=0}^{p-1} \alpha_i \bar{y}(t_{n-i}) + \beta h f(\bar{y}(t_{n+1})) \right) \right).$$

Il est connu que le schéma p-BDF est d'ordre p (voir [SM03]) et qu'il existe une constante $C_p > 0$ tel que pour tout $p-1 \leq n \leq N-1$, $|\tilde{\tau}_n| \leq C_p M_{p+1}(\bar{y})\, h^p$.

De plus, grâce à (2.23), on a $\mathcal{P}_{p,n}(\bar{\boldsymbol{y}}) \in J_p$ et comme f est une fonction Lispchitzienne sur J_p de constante L_p, on en déduit que,

$$|f(\bar{y}(t_{n+1})) - f(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}}))| \leq L_p |\bar{y}(t_{n+1}) - \mathcal{P}_{p,n}(\bar{\boldsymbol{y}})|. \tag{2.42}$$

Alors on en tire que,

$$\begin{aligned} |f(\bar{y}(t_{n+1})) - f(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) - \mathcal{A}_n(\bar{\boldsymbol{y}})(\bar{y}(t_{n+1}) - \mathcal{P}_{p,n}(\bar{\boldsymbol{y}}))| & \\ \leq (L_p + |\mathcal{A}_n(\bar{\boldsymbol{y}})|)\, |\bar{y}(t_{n+1}) - \mathcal{P}_{p,n}(\bar{\boldsymbol{y}}))| & \\ \leq M_p(\bar{y})(L_p + |\mathcal{A}_n(\bar{\boldsymbol{y}})|)\, h^p, & \end{aligned}$$

où nous avons utilisé à nouveau (2.23) pour la dernière inégalité. Cependant soit $\mathcal{A}_n(\bar{\boldsymbol{y}}) = f'(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}}))$ et alors $|\mathcal{A}_n(\bar{\boldsymbol{y}})| \leq L_p$ comme $\mathcal{P}_{p,n}(\bar{\boldsymbol{y}}) \in J_p$ ou soit $\mathcal{A}_n(\bar{\boldsymbol{y}}) = f'(c)$ pour un certain $c \in \mathfrak{E}_f$ et alors $|\mathcal{A}_n(\bar{\boldsymbol{y}})| \leq \sup_{c \in \mathfrak{E}_f} |f'(c)|$. Ainsi, de (2.41), on obtient pour tout $p-1 \leq n \leq N-1$, $|\tau_n| \leq C_1\, h^p$, avec $C_1 = C_p M_{p+1}(\bar{y}) + M_p(\bar{y}) \left(L_p + \sup_{c \in \mathfrak{E}_f} |f'(c)| \right)$ ce qui conclut la preuve. □

2.2.5 Convergence d'ordre p du schéma p-LIBDF

On commence avec la définition suivante de la convergence d'ordre p d'un schéma :

Définition 2.14. La convergence d'ordre p signifie que pour des approximations des points de départs $y_0, y_1, ..., y_{p-1}$ suffisamment précises, l'erreur globale satisfait

$$\max_{0\leq i\leq N} |y_i - \bar{y}(t_i)| = \mathbf{C}h^p, \tag{2.43}$$

avec $\mathbf{C} > 0$ une constante qui peut dépendre de T et $\bar{y}$.

On a alors la proposition suivante :

Proposition 2.15. *La convergence du schéma p-LIBDF est d'ordre p.*

Démonstration. On a vu dans la section 1.1.3 du Chapitre 1 que grâce aux Propositions 2.7 and 2.13, on a la convergence d'ordre p pour le schéma p-LIBDF en utilisant les arguments utilisés dans la preuve du Théorème de Dahlquist (voir Théorème 6.3.4 de [Gau97] ou le Théorème 5.10 de [Hen62]).

□

2.2.6 Majoration de l'erreur globale du schéma LIBDF

Dans la plupart des preuves existantes (voir également la section 1.1.2 du Chapitre 1), la constante symbolisée par la notation $O(h^p)$ est proportionnelle à e^{LT}, où T est la longueur de l'intervalle de simulation et L est une constante de Lipschitz de la fonction f (voir [SM03] p. 318, [BF11] p. 271 et [Pet82] p. 41).

Cependant cette constante est grandement améliorée dans certaines situations, cela est le cas dans cette section, sous l'hypothèse de l'existence d'un état d'équilibre stable. Plus précisément, on montre dans le Théorème 2.20, que quel que soit le point d'équilibre stable $c \in \mathfrak{S}_f$, il existe une boule $\mathcal{B}$ centrée en c telle que si les données initiales $\{y_0, y_1, ..., y_{p-1}\}$ appartiennent à $\mathcal{B}$, alors la convergence du schéma p-LIBDF avec ces données initiales et $\mathcal{A}_n(\boldsymbol{y}) = f'(c)$ est d'ordre p (voir Définition 2.14) avec une constante $\mathbf{C}$ indépendante de T et qui n'explose pas avec la meilleure constante de Lipschitz ou lorsque la raideur du problème devient de plus en plus grande.

Afin de démontrer le Théorème 2.20, on démontre une série de Lemmes et une Proposition (Lemmes 2.17, 2.19 et la Proposition 2.7).

Dans le Lemme 2.17, on montre que pour tout point d'équilibre stable $c \in \mathfrak{S}_f$, il existe une boule $\mathcal{B}_0$ centrée sur c avec un rayon R_μ dépendant de $\mu = \beta h|f'(c)|$ tel que si les données initiales $\{y_0, y_1, ..., y_{p-1}\}$ appartiennent à $\mathcal{B}_1 \subset \mathcal{B}_0$, alors toute la suite numérique $\{y_n\}_{n\in[\![0,N]\!]}$ construite à partir du schéma p-LIBDF avec ces données initiales et $\mathcal{A}_n(\boldsymbol{y}) = f'(c)$ reste dans cette boule $\mathcal{B}_1$.

Dans la Proposition 2.18, sous l'hypothèse que $\sup_{\lambda>0} \alpha_{p,\lambda} < +\infty$, on montre qu'en fait le rayon R_μ admet une borne inférieure indépendante de μ et qu'il existe une boule $\mathcal{B}_2$ dont le rayon est indépendant de μ tel que si les données initiales $\{y_0, y_1, ..., y_{p-1}\}$ appartiennent à $\mathcal{B}_2$, alors toute la suite numérique $\{y_n\}_{n\in[\![0,N]\!]}$ reste dans cette boule $\mathcal{B}_2$.
Dans le Lemme 2.19, on montre que la solution continue $\bar{y}$ de (2.1) est aussi contenue dans une boule $\mathcal{B}_3$ centrée sur c dès que sa condition initiale $y(0) = y_0$ appartient à $\mathcal{B}_3$.

Dans ce qui suit, on a besoin d'introduire $\alpha_{p,\mu} > 0$, le réel défini par

$$\alpha_{p,\mu} = \sup\left\{\frac{\|x\|_\infty \|e_1\|_\mu}{\|x\|_\mu}; x \in \mathbb{R}^p, x \neq 0\right\}, \tag{2.44}$$

la norme $\|\cdot\|_{\infty,\mu}$ définie par $\|\cdot\|_{\infty,\mu} := \dfrac{\|\cdot\|_\mu}{\|e_1\|_\mu}$ et d'utiliser une série de Lemmes.

Grâce à (2.44), on peut également remarquer que pour tout $x \in \mathbb{R}^p$,

$$\|x\|_\infty \leq \alpha_{p,\mu}\|x\|_{\infty,\mu}. \tag{2.45}$$

On note alors $\mathcal{R}(\mu) := \rho_v(A_p(\mu))$. On commence alors avec le Lemme suivant.

Lemme 2.16. *La suite $\{u_n\}_{n\in\mathbb{N}^*}$ définie par $u_{n+1} \leq a_n u_n + b$, $u_0 = 0$ est majorée pour $n \geq 1$ par :*

$$u_n \leq \left(\sum_{k=1}^{n}\prod_{l=k}^{n-1} a_l\right) b. \tag{2.46}$$

Démonstration. La démonstration se déduit facilement par récurrence. □

Lemme 2.17. *Soit c un zéro de la fonction f tel que $f'(c) < 0$, $\{y_n\}_{n\in[\![0,N]\!]}$ satisfaisant le schéma p-LIBDF (2.21) avec les données initiales $\{y_0, y_1, ..., y_{p-1}\}$ et $\mathcal{A}_n(\boldsymbol{y}) = f'(c)$. Soit $\mu = \beta h|f'(c)| \in \mathfrak{D}_p$. Il existe $R > 0$ dépendant seulement de p, μ, f', f'', tel que si $\|\mathcal{D}_{p-1}\|_{\infty,\mu} \leq R$ alors on a pour tout $p \leq n \leq N$, $\|\mathcal{D}_n\|_{\infty,\mu} \leq R$, avec*

$$\mathcal{D}_n = \begin{pmatrix} y_n - c \\ y_{n-1} - c \\ ... \\ y_{n+1-p} - c \end{pmatrix}. \tag{2.47}$$

Le réel $R > 0$ satisfait $R \geq R_\mu$ avec

$$R_\mu = \begin{cases} \inf J \quad avec\ J := \left\{x \in \mathbb{R}^+; g(x) \geq \frac{2|f'(c)|}{\nu_{p,\mu}}(1+\mu)\left(\frac{1-\mathcal{R}(\mu)}{\mu}\right)\right\} si J \neq \emptyset \\ +\infty \quad sinon \end{cases} \tag{2.48}$$

où la fonction g est définie pour tout $x \geq 0$, par :

$$g(x) = x \sup_{z\in I_x} |f''(z)|,$$

avec $I_x = \left[c - x\sqrt{\nu_{p,\mu}}, c + x\sqrt{\nu_{p,\mu}}\right]$ et $\nu_{p,\mu} = (2^p - 1)^2\alpha_{p,\mu}^2$.

Démonstration. Le schéma p-LIBDF est défini pour tout $p - 1 \leq n \leq N - 1$ par

$$y_{n+1} = \sum_{k=0}^{p-1} \alpha_k y_{n-k} + h\beta(f(\mathcal{P}_{p,n}(\boldsymbol{y})) + f'(c)(y_{n+1} - \mathcal{P}_{p,n}(\boldsymbol{y}))). \tag{2.49}$$

Comme $f(c) = 0$ et $\sum_{k=0}^{p-1} \alpha_k = 1$, alors de (2.49), on en déduit

$$y_{n+1} - c = \sum_{k=0}^{p-1} \alpha_k(y_{n-k} - c) + h\beta(f(\mathcal{P}_{p,n}(\boldsymbol{y})) - f(c) + f'(c)((y_{n+1} - c) - (\mathcal{P}_{p,n}(\boldsymbol{y})) - c)). \tag{2.50}$$

En utilisant un développement de Taylor d'ordre deux, on en déduit qu'il existe $\theta_n \in (c, \mathcal{P}_{p,n}(\boldsymbol{y}))$ tel que

$$f(\mathcal{P}_{p,n}(\boldsymbol{y})) - f(c) + f'(c)((y_{n+1} - c) - (\mathcal{P}_{p,n}(\boldsymbol{y})) - c) = \frac{f''(\theta_n)}{2}(\mathcal{P}_{p,n}(\boldsymbol{y})) - c)^2. \quad (2.51)$$

On introduit $\boldsymbol{c} = \{c_i\}_{i\in[\![0,N]\!]}$ avec $c_i = c$. On remarque que $\sum_{k=0}^{p-1}(-1)^k C_p^{k+1} = 1$, en effet $\sum_{k=0}^{p-1}(-1)^k C_p^{k+1} = \sum_{\ell=1}^{p}(-1)^{\ell-1} C_p^{\ell} = -\sum_{\ell=0}^{p}(-1)^{\ell} C_p^{\ell} + 1 = -(1 + (-1))^p + 1 = 1$. Alors, grâce à (2.22), on tire que

$$c = \mathcal{P}_{p,n}(\boldsymbol{c}). \quad (2.52)$$

En posant $\boldsymbol{d} = \{d_i\}_{i\in[\![0,N]\!]}$ avec $d_i = y_i - c$ et en utilisant (2.51), (2.52), de (2.50) on déduit

$$d_{n+1} = \sum_{k=0}^{p-1} \frac{\alpha_k}{1 - \beta h f'(c)} d_{n-k} + \frac{h\beta f''(\theta_n)}{2(1 - h\beta f'(c))} \mathcal{P}_{p,n}(\boldsymbol{d})^2.$$

Ainsi, après avoir posé $f_n = \frac{f''(\theta_n)}{2}\mathcal{P}_{p,n}(\boldsymbol{d})^2$ et $\mu = \beta h|f'(c)|$, on obtient

$$\mathcal{D}_{n+1} = A_p(\mu)\mathcal{D}_n + F_n(\mu), \quad (2.53)$$

avec

$$\mathcal{D}_n = \begin{pmatrix} d_n \\ d_{n-1} \\ \dots \\ d_{n+1-p} \end{pmatrix}, F_n = \begin{pmatrix} \frac{\beta h f_n}{1+\mu} \\ 0 \\ \dots \\ 0 \end{pmatrix}. \quad (2.54)$$

On prend la norme $\|\cdot\|_\mu$ de l'équation (2.53) pour obtenir

$$\|\mathcal{D}_{n+1}\|_\mu \leq \||A_p(\mu)\|\|_\mu \, \|\mathcal{D}_n\|_\mu + \|F_n(\mu)\|_\mu. \quad (2.55)$$

On remarque que $\mathcal{R}(\mu)$ peut également être donné par $\mathcal{R}(\mu) = \||A_p(\mu)\|\|_\mu$ et on remarque que $\|F_n(\mu)\|_\mu = \dfrac{\beta h|f_n|}{1+\mu}\|e_1\|_\mu$. On réécrit (2.55) comme suit

$$\|\mathcal{D}_{n+1}\|_\mu \leq \mathcal{R}(\mu) \, \|\mathcal{D}_n\|_\mu + \frac{\beta h|f_n|}{1+\mu}\|e_1\|_\mu. \quad (2.56)$$

Donnons maintenant une estimation de f_n. On remarque grâce à (2.22) que $|\mathcal{P}_{p,n}(\boldsymbol{d})| \leq \gamma_p \|\mathcal{D}_n\|_\infty$ et grâce à (2.45), il en découle

$$|\mathcal{P}_{p,n}(\boldsymbol{d})| \leq \gamma_p \alpha_{p,\mu} \, \|\mathcal{D}_n\|_{\infty,\mu}. \quad (2.57)$$

Ainsi, grâce à (2.57) on a $|f_n| \leq \frac{\nu_{p,\mu}}{2}|f''(\theta_n)| \, \|\mathcal{D}_n\|^2_{\infty,\mu}$ avec $\nu_{p,\mu} = \gamma_p^2 \alpha_{p,\mu}^2$. Alors, de (2.56), on a

$$\|\mathcal{D}_{n+1}\|_{\infty,\mu} \leq \mathcal{R}(\mu) \, \|\mathcal{D}_n\|_{\infty,\mu} + \frac{\nu_{p,\mu}\beta h}{2(1+\mu)}|f''(\theta_n)| \, \|\mathcal{D}_n\|^2_{\infty,\mu}. \quad (2.58)$$

Comme $c = \mathcal{P}_{p,n}(\boldsymbol{c})$, alors $\mathcal{P}_{p,n}(\boldsymbol{y}) = \mathcal{P}_{p,n}(\boldsymbol{y}) - c + c = \mathcal{P}_{p,n}(\boldsymbol{y} - \boldsymbol{c}) + c = \mathcal{P}_{p,n}(\boldsymbol{d}) + c$ et grâce à nouveau à (2.57) cela nous donne que

$\theta_n \in (c, c+\mathcal{P}_{p,n}(\boldsymbol{d})) \subset I_n := \Big[c-\sqrt{\nu_{p,\mu}}\|\mathcal{D}_n\|_{\infty,\mu}, c+\sqrt{\nu_{p,\mu}}\|\mathcal{D}_n\|_{\infty,\mu}\Big]$. Ainsi, on en déduit que $|f''(\theta_n)| \leq \sup_{z\in I_n}|f''(z)|$. Ainsi, de (2.58) on a

$$\|\mathcal{D}_{n+1}\|_{\infty,\mu} \leq \mathcal{R}(\mu)\,\|\mathcal{D}_n\|_{\infty,\mu} + \frac{\nu_{p,\mu}\beta h}{2(1+\mu)}\sup_{z\in I_n}|f''(z)|\,\|\mathcal{D}_n\|^2_{\infty,\mu}. \tag{2.59}$$

Soit $R>0$ tel que $R \geq \|\mathcal{D}_{p-1}\|_{\infty,\mu}$.
On prouve par récurrence la proposition suivante $\mathcal{P}(n)$: pour tout $p-1 \leq n \leq N$,

$$\|\mathcal{D}_n\|_{\infty,\mu} \leq R.$$

La proposition $\mathcal{P}(n)$ est vraie pour $n=p-1$. Supposons que pour $n \in [\![p-1, N-1]\!]$, la Proposition $\mathcal{P}(n)$ est vraie, alors de (2.59), il est déduit

$$\|\mathcal{D}_{n+1}\|_{\infty,\mu} \leq \mathcal{R}(\mu)\,R + \frac{\nu_{p,\mu}\beta h}{2(1+\mu)}\sup_{z\in I_n}|f''(z)|\,R^2. \tag{2.60}$$

Aussi pour obtenir

$$\|\mathcal{D}_{n+1}\|_{\infty,\mu} \leq R, \tag{2.61}$$

grâce à (2.60), il suffit d'avoir

$$\mathcal{R}(\mu)\,R + \frac{\beta h\nu_{p,\mu}}{2(1+\mu)}R^2\sup_{z\in I_R}|f''(z) \leq R,$$

avec $I_R = \Big[c-\sqrt{\nu_{p,\mu}}R, c+\sqrt{\nu_{p,\mu}}R\Big]$, ce qui signifie également que

$$\frac{\beta h\nu_{p,\mu}}{2(1+\mu)}R\sup_{z\in I_R}|f''(z) \leq 1-\mathcal{R}(\mu). \tag{2.62}$$

Comme $\mu=\beta h|f'(c)|$, alors on a $\beta h = \frac{\mu}{|f'(c)|}$, et ainsi l'inégalité (2.62) est réécrite

$$R\sup_{z\in I_R}|f''(z)| \leq \frac{2|f'(c)|}{\nu_{p,\mu}}(1+\mu)\left(\frac{1-\mathcal{R}(\mu)}{\mu}\right). \tag{2.63}$$

Ainsi, si (2.63) est vérifiée, on obtient alors (2.61). On considère la fonction g définie pour tout $x \geq 0$, par :

$$g(x) = x\sup_{z\in I_x}|f''(z)|,$$

avec $I_x = \Big[c-x\sqrt{\nu_{p,\mu}}, c+x\sqrt{\nu_{p,\mu}}\Big]$. Comme g est continue et $g(0)=0$, alors la valeur R_μ, définie par $R_\mu = \inf\Big\{x\in\mathbb{R}^+; g(x) \geq \frac{2|f'(c)|}{\nu_{p,\mu}}(1+\mu)\Big(\frac{1-\mathcal{R}(\mu)}{\mu}\Big)\Big\}$ est strictement positive et peut éventuellement être égale à $+\infty$ dans le cas où l'ensemble est vide.

En imposant que les données initiales $y_0, y_1, ..., y_{p-1}$ sont telles que $\|\mathcal{D}_{p-1}\|_{\infty,\mu} \leq R_\mu$, alors en prenant $R=R_\mu$, l'inégalité (2.63) est vérifiée et ainsi (2.61) est également vérifiée ce qui signifie que la proposition $\mathcal{P}(n+1)$ est vraie. Ainsi, on en déduit que pour tout $p-1 \leq n \leq N$,

$$\|\mathcal{D}_n\|_{\infty,\mu} \leq R,$$

ce qui conclut la preuve. □

Proposition 2.18. *Soit c un zéro de f tel que $f'(c) < 0$. Soit $\mu = \beta h|f'(c)| \in \mathfrak{D}_p$, $\mu > 0$. Sous l'hypothèse qu'il existe $\kappa_p > 0$ dépendant seulement de p tel que $\sup_{\lambda>0} \alpha_{p,\lambda} \leq \kappa_p$ (voir 2.44 pour $\alpha_{p,\lambda}$) il existe $\eta_p > 0$ dépendant seulement de p tel que le réel R_μ défini dans (2.48) satisfasse $R_\mu \geq R_p$ avec $R_p > 0$ le réel défini par*

$$R_p = \begin{cases} \inf \tilde{J} & \text{avec } \tilde{J} := \{x \in \mathbb{R}^+; g(x) \geq \eta_p |f'(c)|\} \text{ si} \tilde{J} \neq \emptyset \\ +\infty & \text{sinon} \end{cases} \tag{2.64}$$

et la fonction $\tilde{g}$ est définie pour tout $x \geq 0$, par :

$$\tilde{g}(x) = x \sup_{z \in Q_x} |f''(z)|,$$

avec $Q_x = [c - x\chi_p, c + x\chi_p]$ et $\chi_p = (2^p - 1)\kappa_p$.

Démonstration. On rappelle que R_μ est défini par :

$$R_\mu = \begin{cases} \inf J & \text{with } J := \left\{x \in \mathbb{R}^+; g(x) \geq \frac{2|f'(c)|}{\nu_{p,\mu}}(1+\mu)\left(\frac{1-\mathcal{R}(\mu)}{\mu}\right)\right\} if J \neq \emptyset \\ +\infty & elsewhere \end{cases} \tag{2.65}$$

et la fonction g est définie pour tout $x \geq 0$, par :

$$g(x) = x \sup_{z \in I_x} |f''(z)|,$$

avec $I_x = \left[c - x\sqrt{\nu_{p,\mu}}, c + x\sqrt{\nu_{p,\mu}}\right]$ et $\nu_{p,\mu} = (2^p - 1)^2 \alpha_{p,\mu}^2$.
Comme $\alpha_{p,\mu} \leq \kappa_p$, on a $\nu_{p,\mu} \leq \varrho_p := (2^p - 1)\kappa_p$, alors on a pour tout $x \geq 0$

$$I_x \subset Q_x := \left[c - x\sqrt{\varrho_p}, c + x\sqrt{\varrho_p}\right]. \tag{2.66}$$

De plus, grâce au Lemme 2.11, il existe $\varsigma_p > 0$ dépendant seulement de p tel que pour tout $\mu \in \mathfrak{D}_p$, $\mu > 0$, on a

$$\mathcal{R}(\mu) \leq \frac{1}{(1 + \varsigma_p \mu)^{\frac{1}{p}}},$$

ce qui nous donne combiné avec le fait que $\alpha_{p,\mu} \leq \kappa_p$

$$\frac{2|f'(c)|}{\nu_{p,\mu}}(1+\mu)\left(\frac{1-\mathcal{R}(\mu)}{\mu}\right) \geq \frac{2|f'(c)|}{(2^p-1)^2\kappa_p^2}\frac{1+\mu}{\mu}\left(1 - \frac{1}{(1+\varsigma_p\mu)^{\frac{1}{p}}}\right). \tag{2.67}$$

On introduit la fonction continue positive G sur $]0, +\infty[$ définie pour tout $\mu > 0$ par :

$$G_p(\mu) := \frac{1+\mu}{\mu}\left(1 - \frac{1}{(1+\varsigma_p\mu)^{\frac{1}{p}}}\right).$$

On remarque que $G_p(\mu) \to \frac{1}{p}\varsigma_p$ quand $\mu \to 0$ et $G(\mu) \to 1$ quand $\mu \to +\infty$. Alors on déduit que $\Upsilon_p := \inf_{\mu>0} G_p(\mu) > 0$. De (2.67), on trouve que,

$$\frac{2|f'(c)|}{\nu_{p,\mu}}(1+\mu)\left(\frac{1-\mathcal{R}(\mu)}{\mu}\right) \geq \frac{2|f'(c)|}{(2^p-1)^2\kappa_p^2}\Upsilon_p. \tag{2.68}$$

Ainsi, grâce à (2.66) et (2.68), de (2.65) on obtient que $R_\mu \geq R_p$, avec $R_p > 0$ un réel dépendant seulement de p, défini par :

$$R_p = \begin{cases} \inf \tilde{J} \quad \text{avec } \tilde{J} := \left\{ x \in \mathbb{R}^+; \tilde{g}(x) \geq \dfrac{2|f'(c)|}{(2^p-1)^2\kappa_p^2}\Upsilon_p. \right\} si \tilde{J} \neq \emptyset \\ +\infty \quad sinon \end{cases} \tag{2.69}$$

et la fonction $\tilde{g}$ est définie pour tout $x \geq 0$, par :

$$\tilde{g}(x) = x \sup_{z \in Q_x} |f''(z)|,$$

ce qui conclut la preuve. □

Lemme 2.19. *Soit c un zéro de f tel que $f'(c) < 0$ et $0 < \vartheta < 1$. Alors il existe une constante $R_\vartheta > 0$ telle que si $|y_0 - c| < R_\vartheta$, alors la solution $\bar{y}$ de (2.1) satisfait pour tout $t \in [0,T]$, $|y(t) - c| \leq |y_0 - c| e^{\vartheta f'(c) t}$.*

Une valeur possible pour R_ϑ est :

$$R_\vartheta = \sup\{R > 0; \forall z \in \mathbb{R}, |z - c| \leq R \Rightarrow f'(z) \leq \vartheta f'(c)\}.$$

Démonstration. Comme $f(c) = 0$, alors de (2.1), il vient que pour tout $t \in [0,T]$,

$$\frac{\mathrm{d}(\bar{y}(t) - c)}{\mathrm{d}t} = f(\bar{y}(t)) - f(c). \tag{2.70}$$

En multipliant l'équation (2.70) par $\bar{y}(t) - c$ cela conduit à

$$\frac{1}{2}\frac{\mathrm{d}|\bar{y}(t) - c|^2}{\mathrm{d}t} = (f(\bar{y}(t)) - f(c))(\bar{y}(t) - c). \tag{2.71}$$

Grâce au développement de Taylor, quel que soit $t \in [0,T]$, il existe $\theta(t) \in (c, y(t))$ tel que

$$f(\bar{y}(t)) - f(c) = f'(\theta(t))(\bar{y}(t) - c).$$

Alors, de (2.71), on obtient

$$\frac{1}{2}\frac{\mathrm{d}|\bar{y}(t) - c|^2}{\mathrm{d}t} = f'(\theta(t))|\bar{y}(t) - c|^2. \tag{2.72}$$

Comme f' est continue et $f'(c) < 0$, alors quel que soit $0 < \vartheta < 1$ il existe $R_\vartheta > 0$ tel que pour tout $z \in \mathbb{R}$ tel que $|z - c| \leq R_\vartheta$, on a $f'(z) \leq \vartheta f'(c) < 0$. y_0 doit alors vérifier $|y_0 - c| < R_\vartheta$.

Montrons que pour tout $t \in [0,T]$, $|y(t) - c| \leq R_\vartheta$. S'il existe $t_0 \in [0,T]$ tel que $|y(t_0) - c| > R_\vartheta$, alors la valeur $t_* = \inf\{t \in [0,T]; |y(t) - c| > R_\vartheta\}$ est bien définie. Suite à la définition de t_* et du fait que y soit continue, on tire que

$$|y(t_*) - c| = R_\vartheta, \tag{2.73}$$

et alors $t_* \neq 0$, comme $t_* > 0$.
De plus, grâce à nouveau à la définition de t_*, on a que pour tout $t \in [0, t_*]$,

$$|y(t) - c| \leq R_\vartheta, \tag{2.74}$$

et comme $\theta(t) \in (c, y(t))$ alors on a également $|\theta(t) - c| \leq R_\vartheta$, ce qui implique que $f'(\theta(t)) \leq \vartheta f'(c) < 0$. Ainsi, à partir de (2.72), on en déduit que pour tout $t \in [0, t_*]$,

$$\frac{1}{2}\frac{d|\bar{y}(t) - c|^2}{dt} \leq \vartheta f'(c)|\bar{y}(t) - c|^2. \tag{2.75}$$

On remarque que

$$\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\left(|\bar{y}(t) - c|^2 e^{-2\vartheta f'(c)t}\right) = e^{-2\vartheta f'(c)t}\left(\frac{1}{2}\frac{\mathrm{d}|\bar{y}(t) - c|^2}{\mathrm{d}t} - \vartheta f'(c)|\bar{y}(t) - c|^2\right).$$

C'est pourquoi l'inégalité (2.75) est réécrite comme,

$$\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\left(|\bar{y}(t) - c|^2 e^{-2\vartheta f'(c)t}\right) \leq 0. \tag{2.76}$$

Alors, de (2.76), on déduit que pour tout $t \in [0, t_*]$,

$$|\bar{y}(t) - c| \leq |y_0 - c|e^{\vartheta f'(c)t}. \tag{2.77}$$

Comme $|y_0 - c| < R_\vartheta$ et $f'(c) < 0$, on obtient que $|\bar{y}(t_*) - c| < R_\vartheta$, ce qui conduit à une contradiction avec (2.73). C'est pourquoi, on peut conclure que pour tout $t \in [0, T]$, $|y(t) - c| \leq R_\vartheta$. De (2.74), en appliquant à T les mêmes arguments utilisés pour t_*, on déduit que pour tout $t \in [0, T]$,

$$|\bar{y}(t) - c| \leq |y_0 - c|e^{\vartheta f'(c)t},$$

ce qui conclut la preuve. □

Maintenant, revenons à la preuve de notre Théorème. Grâce à la Proposition 2.13 et (2.15), la troisième condition dans le Théorème 2.20 (avec τ défini dans la Définition 2.12)) est exprimée comme une condition sur h et indépendante du temps de simulation T. On désigne par E_n l'erreur globale au temps t_n. Pour simplifier, on suppose qu'il n'y a pas d'erreur sur les données initiales (cela signifie que $E_{p-1} = 0$)

Théorème 2.20. *Soit c un zéro de f tel que $f'(c) < 0$. Soit $\mu = \beta|f'(c)|h \in \mathfrak{D}_p$ et $0 < \vartheta < 1$. Sous l'hypothèse qu'il existe $\kappa_p > 0$ dépendant seulement de p tel que $\sup_{\lambda>0} \alpha_{p,\lambda} \leq \kappa_p$ (voir 2.44 pour $\alpha_{p,\lambda}$), il existe $R_p > 0$ dépendant uniquement de p, $R_\vartheta > 0$ dépendant uniquement de ϑ, $\Theta_{p,\vartheta} > 0, \Phi_{p,\vartheta} > 0$ et $\Psi_{p,\vartheta}$ dépendant uniquement de p, ϑ tel que si*

$$\begin{aligned} \|\mathcal{D}_{p-1}\|_{\infty,\mu} &\leq R_p, \\ |y_0 - c| &\leq R_\vartheta, \\ \tau &\leq \Theta_{p,\vartheta}\, e^{-2\frac{\Psi_{p,\vartheta}}{|f'(c)|}}|f'(c)|^2, \\ E_{p-1} &= 0, \end{aligned}$$

alors pour tout $p - 1 \leq n \leq N - 1$, $E_n \leq \frac{\Phi_{p,\vartheta}\, e^{\frac{\Psi_{p,\vartheta}}{|f'(c)|}}}{|f'(c)|}\tau$. Pour tout $p - 1 \leq n \leq N - 1$, $E_n := \|\mathcal{E}_n\|_{\infty,\mu}$ avec,

$$\mathcal{E}_n = \begin{pmatrix} y_n - \bar{y}(t_n) \\ y_{n-1} - \bar{y}(t_{n-1}) \\ \dots \\ y_{n+1-p} - \bar{y}(t_{n+1-p}) \end{pmatrix} \text{ et } \mathcal{D}_{p-1} = \begin{pmatrix} y_{p-1} - c \\ y_{p-2} - c \\ \dots \\ y_0 - c \end{pmatrix}. \tag{2.78}$$

Démonstration. Le schéma p-LIBDF est le suivant : pour tout $p-1 \leq n \leq N-1$,

$$y_{n+1} = \sum_{k=0}^{p-1} \alpha_k y_{n-k} + h\beta(f(\mathcal{P}_{p,n}(\boldsymbol{y})) + f'(c)(y_{n+1} - \mathcal{P}_{p,n}(\boldsymbol{y}))). \tag{2.79}$$

La définition de l'erreur de troncature du schéma p-LIBDF conduit à,

$$\bar{y}(t_{n+1}) = \sum_{k=0}^{p-1} \alpha_k \bar{y}(t_{n-k}) + \beta h(f(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) + f'(c)(\bar{y}(t_{n+1}) - \mathcal{P}_{p,n}(\bar{\boldsymbol{y}}))) + h\tau_n, \tag{2.80}$$

avec $\bar{\boldsymbol{y}} = \{\bar{y}(t_i)\}_{i\in[\![0,N]\!]}$.

En soustrayant (2.79) et (2.80), et en utilisant le fait que l'application $\mathcal{P}_{p,n}(\cdot)$ est linéaire, on observe que

$$\epsilon_{n+1} = \sum_{k=0}^{p-1} \alpha_k \epsilon_{n-k} + \beta h(f(\mathcal{P}_{p,n}(\boldsymbol{y})) - f(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) + f'(c)(\epsilon_{n+1} - \mathcal{P}_{p,n}(\boldsymbol{\epsilon}))) - h\tau_n, \tag{2.81}$$

avec $\boldsymbol{\epsilon} = \{\epsilon_i\}_{i\in[\![0,N]\!]}$ et $\epsilon_i = y_i - \bar{y}(t_i)$. En utilisant un développement de Taylor d'ordre 2, on déduit qu'il existe $\theta_n \in (\mathcal{P}_{p,n}(\boldsymbol{y}), \mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) \subset (0,T)$ tel que

$$f(\mathcal{P}_{p,n}(\boldsymbol{y})) = f(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) + f'(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}}))\mathcal{P}_{p,n}(\boldsymbol{\epsilon}) + \frac{f''(\theta_n)}{2}\mathcal{P}_{p,n}(\boldsymbol{\epsilon})^2. \tag{2.82}$$

En insérant l'expression (2.82) dans (2.81), il vient :

$$\begin{aligned}(1-\beta h f'(c))\epsilon_{n+1} &= \sum_{k=0}^{p-1} \alpha_k \epsilon_{n-k} + \beta h(f'(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) - f'(c))\mathcal{P}_{p,n}(\boldsymbol{\epsilon}) \\ &\quad + \beta h \frac{f''(\theta_n)}{2}\mathcal{P}_{p,n}(\boldsymbol{\epsilon})^2 - h\tau_n.\end{aligned} \tag{2.83}$$

En utilisant un développement de Taylor d'ordre un, on déduit qu'il existe $\beta_n \in (c, \mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) \subset (0,T)$ tel que

$$f'(\mathcal{P}_{p,n}(\bar{\boldsymbol{y}})) - f'(c) = f''(\beta_n)\mathcal{P}_{p,n}(\bar{\boldsymbol{y}} - \boldsymbol{c}), \tag{2.84}$$

avec $\boldsymbol{c} = \{c_i\}_{i\in[\![0,N]\!]}$, et $c_i = c$.

En insérant l'expression (2.84) dans (2.83), on en déduit

$$\begin{aligned}(1-\beta h f'(c))\epsilon_{n+1} &= \sum_{k=0}^{p-1} \alpha_k \epsilon_{n-k} + \beta h f''(\beta_n)\mathcal{P}_{p,n}(\bar{\boldsymbol{y}} - \boldsymbol{c})\mathcal{P}_{p,n}(\boldsymbol{\epsilon}) \\ &\quad + \beta h \frac{f''(\theta_n)}{2}\mathcal{P}_{p,n}(\boldsymbol{\epsilon})^2 - h\tau_n.\end{aligned} \tag{2.85}$$

On pose

$$\phi_n = \beta h f''(\beta_n)\mathcal{P}_{p,n}(\bar{\boldsymbol{y}} - \boldsymbol{c})\mathcal{P}_{p,n}(\boldsymbol{\epsilon}) + \beta h \frac{f''(\theta_n)}{2}\mathcal{P}_{p,n}(\boldsymbol{\epsilon})^2. \tag{2.86}$$

On tire alors que

$$(1 - h\beta f'(c))\epsilon_{n+1} = \sum_{k=0}^{p-1} \alpha_k \epsilon_{n-k} + \phi_n - h\tau_n,$$

ce qui implique que

$$\epsilon_{n+1} = \sum_{k=0}^{p-1} \frac{\alpha_k}{1 - h\beta f'(c)} \epsilon_{n-k} + \frac{\phi_n - h\tau_n}{1 - h\beta f'(c)}.$$

Ainsi, en posant $\mu = \beta h |f'(c)|$, on obtient

$$\mathcal{E}_{n+1} = A_p(\mu)\mathcal{E}_n + B_n, \tag{2.87}$$

avec

$$\mathcal{E}_n = \begin{pmatrix} \epsilon_n \\ \epsilon_{n-1} \\ \dots \\ \epsilon_{n+1-p} \end{pmatrix}, B_n = \begin{pmatrix} \frac{\phi_n - h\tau_n}{1+\mu} \\ 0 \\ \dots \\ 0 \end{pmatrix}. \tag{2.88}$$

On prend la norme $\|\cdot\|_\mu$ de l'équation (2.87) pour obtenir

$$\|\mathcal{E}_{n+1}\|_\mu \leq |||A_p(\mu)|||_\mu \, \|\mathcal{E}_n\|_\mu + \|B_n\|_\mu. \tag{2.89}$$

De (2.86), on en déduit

$$|\phi_n| \leq \beta h K_n \left(|\mathcal{P}_{p,n}(\bar{\boldsymbol{y}} - \boldsymbol{c})| \, |\mathcal{P}_{p,n}(\boldsymbol{\epsilon})| + \frac{1}{2} \mathcal{P}_{p,n}(\boldsymbol{\epsilon})^2 \right),$$

avec $K_n = \max(|f''(\theta_n)|, |f''(\beta_n)|)$.

Grâce à (2.22) avec $\gamma_p = 2^p - 1$, on en déduit,

$$|\mathcal{P}_{p,n}(\boldsymbol{\epsilon})| \leq \gamma_p \|\mathcal{E}_n\|_\infty. \tag{2.90}$$

On obtient alors

$$|\phi_n| \leq \beta h K_n (\gamma_p |\mathcal{P}_{p,n}(\bar{\boldsymbol{y}} - \boldsymbol{c})| \|\mathcal{E}_n\|_\infty + 0.5 \gamma_p^2 \|\mathcal{E}_n\|_\infty^2).$$

Grâce à (2.45), après avoir posé $\nu_p := \gamma_p \kappa_p \geq \gamma_p \alpha_{p,\mu}$, on en déduit

$$|\phi_n| \leq \beta h K_n (\nu_p |\mathcal{P}_{p,n}(\bar{\boldsymbol{y}} - \boldsymbol{c})| \|\mathcal{E}_n\|_{\infty,\mu} + 0.5 \nu_p^2 \|\mathcal{E}_n\|_{\infty,\mu}^2).$$

De plus, on remarque que $\|B_n\| = \frac{|\phi_n - h\tau_n|}{1+\mu} \|e_1\|_\mu$. On pose $E_n = \|\mathcal{E}_n\|_{\infty,\mu}$ et $\mathcal{R}(\mu) = |||A_p(\mu)|||_\mu$, alors de (2.89), on obtient.

$$E_{n+1} \leq \left(\mathcal{R}(\mu) + \frac{\beta h K_n \nu_p}{1+\mu} |\mathcal{P}_{p,n}(\bar{\boldsymbol{y}} - \boldsymbol{c})| \right) E_n + \frac{\beta h K_n \nu_p^2}{2(1+\mu)} E_n^2 + \frac{h\tau}{1+\mu}, \tag{2.91}$$

avec $\tau = \max_{n \in [\![p-1, N-1]\!]} |\tau_n|$. Grâce au Lemme 2.17 et à la Proposition 2.18, il existe $R_p > 0$ dépendant seulement de p tel que

$$\|\mathcal{D}_{p-1}\|_{\infty,\mu} \leq R_p \tag{2.92}$$

implique pour tout $p \leq n \leq N$

$$\|\mathcal{D}_n\|_{\infty,\mu} \leq R_p, \tag{2.93}$$

avec pour tout $p-1 \leq n \leq N$,

$$\mathcal{D}_n = \begin{pmatrix} y_n - c \\ y_{n-1} - c \\ \dots \\ y_{n+1-p} - c \end{pmatrix}. \tag{2.94}$$

Grâce à (2.45), on a

$$\|\mathcal{D}_n\|_\infty \leq \alpha_{p,\mu}\|\mathcal{D}_n\|_{\infty,\mu} \leq \kappa_p \|\mathcal{D}_n\|_{\infty,\mu}. \tag{2.95}$$

De plus, grâce au Lemme 2.19, il existe $R_\vartheta > 0$ dépendant seulement de ϑ tel que

$$|y_0 - c| \leq R_\vartheta \tag{2.96}$$

implique pour tout $t \in [0,T]$,

$$|\bar{y}(t) - c| \leq |y_0 - c| e^{\vartheta f'(c)t}. \tag{2.97}$$

Dans ce qui suit, on suppose que (2.92) et (2.96) sont vérifiées. Alors (2.93) et (2.93) sont également vérifiées et grâce à (2.95), on tire qu'il existe $R_{p,\vartheta} > 0$ ne dépendant que de p, ϑ tel que pour tout $p-1 \leq n \leq N-1$,

$$K_n \leq K_{p,\vartheta} := \sup_{z \in [c-R_{p,\vartheta}, c+R_{p,\vartheta}]} |f''(z)|.$$

Grâce à (2.97), on obtient $|\mathcal{P}_{p,n}(\bar{\boldsymbol{y}} - \boldsymbol{c})| \leq z_p(t_n) := \zeta_p \exp(-\vartheta |f'(c)| t_{n-p})$ avec $\zeta_p = (2^p - 1)|y_0 - c|$.

L'inégalité (2.91) devient

$$E_{n+1} \leq \left(\mathcal{R}(\mu) + \frac{\beta h K_{p,\vartheta} \nu_p}{1+\mu} z_p(t_n) \right) E_n + \frac{\beta h K_{p,\vartheta} \nu_p^2}{2(1+\mu)} E_n^2 + \frac{h\tau}{1+\mu}. \tag{2.98}$$

On pose $\epsilon_0 = \dfrac{h\tau}{1+\mu}$. On prouve par récurrence la proposition suivante :

$$\forall n, \quad p-1 \leq n \leq N, \qquad E_n \leq \theta \epsilon_0, \tag{2.99}$$

avec $\theta > 1$ satisfaisant (2.103). On suppose que $E_{p-1} \leq \theta\tau$. Alors la proposition est vraie pour $n = p-1$. On suppose que la proposition est vraie jusqu'au rang n et nous montrons qu'elle est vraie au rang $n+1$.

En utilisant l'hypothèse de récurrence, on a :

$$E_{n+1} \leq \left(\mathcal{R}(\mu) + \frac{\beta h K_{p,\vartheta} \nu_p}{1+\mu} z_p(t_n) \right) E_n + \frac{\beta h K_{p,\vartheta} \nu_p^2}{2(1+\mu)} \theta^2 \epsilon_0^2 + \epsilon_0. \tag{2.100}$$

On pose

$$b = \epsilon_0 \left(1 + \frac{\beta h K_{p,\vartheta} \nu_p^2}{2(1+\mu)} \theta^2 \epsilon_0 \right). \tag{2.101}$$

.

On utilise le Lemme 2.16 pour obtenir :

$$E_{n+1} \leq \left(\sum_{k=1}^{n+1} \prod_{l=k}^{n} \left(\mathcal{R}(\mu) + \frac{\beta h K_{p,\vartheta} \nu_p}{1+\mu} z_p(t_l) \right) \right) b.$$

Grâce au Lemme 2.11, il existe $\varrho_p > 0$ ne dépendant que de p tel que

$$\mathcal{R}(\mu) \leq \mathcal{G}(\mu) := \frac{1}{(1+\varrho_p\mu)^{\frac{1}{p}}}. \tag{2.102}$$

On a alors,

$$\begin{aligned}
E_{n+1} &\leq \left(\sum_{k=1}^{n+1}\prod_{l=k}^{n}\left(\mathcal{G}(\mu)+\frac{\beta h K_{p,\vartheta}\nu_p}{1+\mu}z_p(t_l)\right)\right)b \\
&\leq \left(\sum_{k=1}^{n+1}\mathcal{G}(\mu)^{n-k+1}\prod_{l=k}^{n}\left(1+\frac{\beta h K_{p,\vartheta}\nu_p}{(1+\mu)\mathcal{G}(\mu)}z_p(t_l)\right)\right)b \\
&\leq \left(\sum_{k=1}^{n+1}\mathcal{G}(\mu)^{n-k+1}\prod_{l=k}^{n}\mathrm{e}^{\frac{\beta h K_{p,\vartheta}\nu_p}{(1+\mu)\mathcal{G}(\mu)}z_p(t_l)}\right)b \\
&= \left(\sum_{k=1}^{n+1}\mathcal{G}(\mu)^{n-k+1}\mathrm{e}^{\sum_{l=k}^{n}\frac{\beta h K_{p,\vartheta}\nu_p}{(1+\mu)\mathcal{G}(\mu)}z_p(t_l)}\right)b.
\end{aligned}$$

Comme $t_l = h\,l$, on observe que $\displaystyle\sum_{l=k}^{n} z_p(t_l) \leq \frac{\zeta_p}{1-e^{-\frac{\vartheta\mu}{\beta}}}$. Ainsi, on a

$$\exp(\sum_{l=k}^{n}\frac{\beta h K_{p,\vartheta}\nu_p}{(1+\mu)\mathcal{G}(\mu)}z_p(t_l)) \leq \omega_{p,\mu,\vartheta} := \exp\left(\frac{K_{p,\vartheta}\zeta_p\nu_p}{(1+\mu)\mathcal{G}(\mu)}\frac{\beta h}{(1-e^{-\frac{\vartheta\mu}{\beta}})}\right).$$

Alors, on obtient

$$\begin{aligned}
E_{n+1} &\leq b\,\omega_{p,\mu,\vartheta}\sum_{k=1}^{n+1}\mathcal{G}(\mu)^{n-k+1} \\
&\leq b\,\frac{\omega_{p,\mu,\vartheta}}{1-\mathcal{G}(\mu)}.
\end{aligned}$$

Comme $\mathcal{G}(\mu) := \frac{1}{(1+\varrho_p\mu)^{\frac{1}{p}}}$, on a $(1+\mu)\mathcal{G}(\mu) \geq \frac{1}{\upsilon_p}(1+\mu)^{1-\frac{1}{p}}$ avec $\upsilon_p = \max(1,\varrho_p)^{\frac{1}{p}}$. Comme la dérivée seconde de $h \longmapsto e^{-\vartheta|f'(c)|h}$ est décroissante, on a que pour tout $h > 0$, $\dfrac{h}{1-e^{-\vartheta|f'(c)|h}} \leq \dfrac{1}{\vartheta|f'(c)|}$

Ainsi, on a

$$\omega_{p,\mu,\vartheta} \leq \psi_{p,\vartheta} := \exp\left(K_{p,\vartheta}\zeta_p\nu_p\upsilon_p\frac{\beta}{\vartheta|f'(c)|}\right),$$

et on obtient $E_{n+1} \leq b\,\dfrac{\psi_{p,\vartheta}}{1-\mathcal{G}(\mu)}$. Dans ce qui suit, on suppose que

$$\theta = \frac{3}{2}\frac{\psi_{p,\vartheta}}{1-\mathcal{G}(\mu)}. \tag{2.103}$$

Après avoir utilisé la valeur de b donnée dans (2.101), on en déduit que pour obtenir $E_{n+1} \leq \theta\epsilon_0$, il suffit d'avoir

$$\left(1+\frac{\beta h K_{p,\vartheta}\nu_p^2}{2(1+\mu)}\theta^2\epsilon_0\right)\frac{\psi_{p,\vartheta}}{1-\mathcal{G}(\mu)} \leq \theta. \tag{2.104}$$

En utilisant la valeur de ϵ_0, l'inégalité (2.104) devient

$$\left(1+\frac{\beta K_{p,\vartheta}\nu_p^2}{2}\theta^2\left(\frac{h}{1+\mu}\right)^2\tau\right)\frac{\psi_{p,\vartheta}}{1-\mathcal{G}(\mu)}\leq\theta. \tag{2.105}$$

L'inégalité (2.105) se réécrit en utilisant (2.103) sous la forme

$$\begin{aligned}\tau &\leq 2\left(\frac{1+\mu}{h}\right)^2\frac{\left(\frac{1-\mathcal{G}(\mu)}{\psi_{p,\vartheta}}\right)\theta-1}{\beta K_{p,\vartheta}\nu_p^2\theta^2}\\ &= \frac{4}{9}\frac{(1+\mu)^2}{\beta K_{p,\vartheta}\nu_p^2\psi_{p,\vartheta}^2}\left(\frac{1-\mathcal{G}(\mu)}{h}\right)^2.\end{aligned} \tag{2.106}$$

Comme $h=\frac{\mu}{\beta|f'(c)|}$ et $\mathcal{G}(\mu):=\frac{1}{(1+\varrho_p\mu)^{\frac{1}{p}}}$, l'inégalité (2.106) devient

$$\tau\leq\frac{4}{9}\frac{\beta|f'(c)|^2}{K_{p,\vartheta}\nu_p^2\psi_{p,\vartheta}^2}\left(\frac{(1+\mu)\left(1-\frac{1}{(1+\varrho_p\mu)^{\frac{1}{p}}}\right)}{\mu}\right)^2. \tag{2.107}$$

On introduit la fonction positive w sur $]0,+\infty[$ définie pour tout $x>0$ par : $w(x)=\frac{(1+x)\left(1-\frac{1}{(1+\varrho_p x)^{\frac{1}{p}}}\right)}{x}$. On remarque que $w(x)\to\frac{1}{p}\varrho_p$ quand $x\to 0^+$ et $w(x)\to 1$ quand $x\to+\infty$, ainsi on en déduit que $\Gamma_p=\inf_{x>0}w(x)>0$. Alors, l'inégalité (2.107) est vérifiée si on a

$$\tau\leq\frac{4}{9}\frac{\beta|f'(c)|^2}{K_{p,\vartheta}\nu_p^2\psi_{p,\vartheta}^2}\Gamma_p^2. \tag{2.108}$$

En exigeant alors que l'inégalité (2.108) soit vérifiée, on obtient que $E_{n+1}\leq\theta\epsilon_0$ et la proposition au rang $n+1$ est vraie. Ainsi, on déduit que pour tout $p-1\leq n\leq N$,

$$E_n\leq\theta\epsilon_0,$$

ce qui donne avec les valeurs de θ et ϵ_0

$$\begin{aligned}E_n &\leq \frac{3}{2}\frac{\psi_{p,\vartheta}}{(1-\mathcal{G}(\mu))}\frac{h\tau}{1+\mu}\\ &= \frac{3}{2\beta|f'(c)|}\frac{\psi_{p,\mu,h}}{(1-\mathcal{G}(\mu))}\frac{\mu}{1+\mu}\tau\\ &= \frac{3\psi_{p,\vartheta}}{2\beta|f'(c)|}\frac{\tau}{w(\mu)}\\ &\leq \frac{3\psi_{p,\vartheta}}{2\beta|f'(c)|}\frac{\tau}{\Gamma_p},\end{aligned}$$

et termine la preuve. □

2.2.7 Choix de l'interpolation

On a vu dans la Section 2.2.2 que $P_{p,n}(t,\boldsymbol{u})$ désigne l'interpolation polynomiale de Lagrange de degré $p-1$ pour les points $\{(t_{n-i},u_{n-i}):i=0,1,..,p-1\}$. Cette interpolation a l'avantage d'être rapide à calculer car il s'agit d'une combinaison linéaire des points

précédents qui sont déjà stockés en mémoire. Cependant, lorsqu'on augmente l'ordre d'interpolation afin de coller à l'ordre du schéma LIBDF, on note l'apparition du phénomène de Runge. Le phénomène de Runge décrit le fait que plus on fixe de points où le polynôme d'interpolation à la même valeur que la fonction à approximer moins bien on approche celle-ci. En effet, si on note l'interpolation $P_{p,n}(t, \boldsymbol{u})$ comme :

$$P_{p,n}(t, \boldsymbol{u}) = \sum_{i=0}^{p} u(t_{n-i}) l_i(t) \tag{2.109}$$

avec (l_i) la base des polynômes de Lagrange liés au temps (t_i), on a alors :

$$|P_{p,n}(t, \boldsymbol{u})| \leq \left(\sum_{i=0}^{p} |l_i(t)| \right) \|u\|_\infty \leq \sup_{x \in [a,b]} \left(\sum_{i=0}^{p} |l_i(t)| \right) \|u\|_\infty. \tag{2.110}$$

La constante $\sup_{x\in[a,b]} \left(\sum_{i=0}^{p} |l_i(t)|\right)$ est estimée dans le cas de pas de temps constant, approximativement égale à (voir Section 2.1 de [Smi06]) :

$$\sup_{x \in [a,b]} \left(\sum_{i=0}^{p} |l_i(t)| \right) \approx \frac{2^{p+1}}{p \ln(p)}. \tag{2.111}$$

On remarque que quand p est grand cette constante augmente plus rapidement que l'interpolation ne peut se rapprocher de u.

Ainsi dans le cas des ordres élevés, pour pallier à cette explosion de la constante de l'erreur, on n'utilise plus des polynômes de Lagrange. En effet, on utilise les racines des polynômes de Tchebychev comme points d'interpolation en effectuant une transformation affine de l'intervalle de simulation.

2.2.8 Méthode de résolution du système linéaire

On a vu dans l'équation (2.20) que le schéma LIBDF requiert l'inverse de la matrice $(I - \beta_n h_n \mathcal{A}_n(\boldsymbol{y}))$. Cependant, en pratique, on ne calcule pas toujours cette matrice inverse. En effet, dans le cas de systèmes linéaires creux, on privilégie l'utilisation de méthodes itératives pour les résoudre. Ainsi, on utilise l'algorithme du gradient biconjugué car, pour la plupart des modèles, la matrice $(I - \beta_n h_n \mathcal{A}_n(\boldsymbol{y}))$ est creuse et l'algorithme du gradient biconjugué convient à la résolution de ce type de matrice.

2.3 Estimation de l'erreur et gestion du pas

On a vu dans la section 1.1.6 du Chapitre 1 que souvent il n'est pas possible de déterminer exactement l'erreur commise, en particulier l'erreur globale. On parvient cependant à estimer l'erreur locale (ou erreur de troncature) en utilisant deux schémas à deux ordres différents comme on a vu dans la section 1.1.6 du Chapitre 1. L'idée dans ce paragraphe est d'estimer l'erreur de la manière la plus économique possible en terme de temps de calcul.

2.3.1 Estimation de l'erreur

Nous avons à notre disposition deux valeurs pour une solution dont nous cherchons à estimer l'erreur :

– la solution donnée par l'interpolation de Lagrange $\mathcal{P}_{p,n}(\boldsymbol{y})$,

– la solution donnée par le schéma LIBDF y_{n+1}.

On rappelle qu'on définit l'erreur locale (ou erreur de troncature) en utilisant $u(t)$ la solution du problème

$$u' = f(t, u), \tag{2.112}$$
$$u(t_n) = y_n, \tag{2.113}$$

et l'erreur locale est alors :

$$le_n = \frac{u(t_n + h_n) - y_{n+1}}{h}. \tag{2.114}$$

L'erreur de troncature pour chacun de ces schémas est :

– pour l'interpolation de Lagrange d'ordre p : $u(t_n + h_n) - y^*_{n+1} = C^*_p f^{(p)}(u)h^p$,

– pour le schéma LIBDF d'ordre p : $u(t_n + h_n) - y_{n+1} = C_{p+1} f^{(p+1)}(u)h^{p+1}$.

En faisant la différence indiquée dans la section 1.1.6 du Chapitre 1, on a alors :

$$y^*_{n+1} - y_{n+1} = [u(t_n + h_n) - y_{n+1}] - [u(t_n + h_n) - y^*_{n+1}] \tag{2.115}$$
$$= C^*_p f^{(p)}(u)h^p + O(h^{p+1}). \tag{2.116}$$

On a alors un estimateur d'ordre p qui estime ici au mieux l'interpolation de Lagrange. Cet estimateur est aussi valable pour le schéma p-LIBDF et est surtout très économique car il ne nécessite pas de calculs supplémentaires.

2.3.2 Calcul du pas de temps

On a vu dans le paragraphe précédent qu'on pouvait estimer l'erreur de la solution issue du schéma numérique par rapport à la solution exacte. Cette estimation permet de calculer le pas de temps maximal que peut utiliser le schéma numérique. En effet, lors d'une simulation, il est fréquent que l'utilisateur du solveur EDO spécifie une tolérance maximale sur l'erreur que produit le schéma numérique. Comme indiqué dans la section 1.1.6 du Chapitre 1, on choisit le pas de temps de manière à ce que l'estimateur soit égal à la tolérance fixée par l'utilisateur. On donne alors α le rapport d'augmentation (ou de diminution) du pas de temps :

$$\alpha = \left(\frac{\tau}{\|est_n\|} \right)^{1/p}. \tag{2.117}$$

2.3.3 Schéma LIBDF en pas variable

On a vu dans la section consacrée au schéma LIBDF que dans le cas où le pas de temps est variable il est nécessaire d'adapter le schéma LIBDF. On a vu dans le paragraphe précédent que la stratégie qui a été choisie dans cette thèse est de modifier les coefficients du schéma LIBDF afin de tenir compte de ce pas variable. Cependant, ce n'est pas la seule manière d'utiliser le schéma LIBDF en pas variable. Il existe trois principales stratégies permettant l'utilisation du schéma LIBDF en pas variable et nous montrons dans ce paragraphe en quoi la stratégie choisie (la modification des coefficients) est la plus rapide et la plus stable pour le schéma LIBDF.

Stratégie à coefficients fixes (Fixed Coefficient Strategy, en abrégé INT)

Il est possible d'utiliser une formule LIBDF avec les coefficients utilisés dans le cas du pas constant. Cette stratégie est appelée Fixed Coefficient Strategy. Prenons par exemple le cas du schéma LIBDF d'ordre 2 à pas variable mais avec les coefficients du pas fixe :

$$y_{n+1} = \frac{4}{3}y_n - \frac{1}{3}y_{n-1} + \frac{2}{3}h_n(f(\mathcal{P}_{p,n}(\boldsymbol{y})) + f'(c)(y_{n+1} - \mathcal{P}_{p,n}(\boldsymbol{y}))). \tag{2.118}$$

Le schéma LIBDF requiert les valeurs de la solution aux temps $t_{n-1} = t_n - h_n$ et $t_{n-1} - h_n$ car les coefficients sont ceux utilisés en pas fixe h_n. Or on dispose seulement des solutions données par le pas variable à savoir, les solutions aux temps t_{n-1}, $t_{n-1} - h_{n-1}$ et $t_{n-1} - h_{n-1} - h_{n-2}$. On utilise alors une interpolation polynomiale pour déterminer les points nécessaires, que l'on utilise dans la formule à coefficients constants. Cette opération est recommencée à chaque pas de temps.

L' avantage de cette stratégie est sa simplicité, car il n'y a pas besoin de déterminer la valeur des coefficients en pas variable et de les évaluer à chaque pas de temps.

Cependant les inconvénients sont nombreux :

- coût de calcul important des interpolations en particulier dans le cas des ordres élevés. En effet à chaque pas de temps, on doit calculer une combinaison linéaire de vecteurs qui sont de taille égale à la taille du système. Dans le cas des systèmes de grande taille, cette combinaison linéaire apporte un surcoût important.
- l'erreur de l'interpolation conduit à d'importantes instabilités, en particulier cette stratégie est moins stable que celle qui consiste à utiliser des coefficients variables. En effet, l'interpolation peut produire des points avec une erreur très importante, en particulier dans le cas où le point à interpoler est très proche d'un point utilisé pour l'interpolation. De plus, le phénomène de Runge entraîne une erreur d'interpolation importante dans le cas des ordres élevés. On verra dans la Section 2.3.3 les conditions de stabilité de cette méthode.

En raison des deux derniers points cette stratégie n'est pas utilisée.

Un compromis entre la stratégie à coefficients fixes et la stratégie à coefficients constants existe, il s'agit de la stratégie dite "Fixed Leading-Coefficient Strategy"

Stratégie du coefficient fixe unique (Fixed Leading Coeffcient Strategy, en abrégé FLC)

On considère tout d'abord le polynôme de Lagrange qui interpole y_{n-i} et que l'on a noté $\mathcal{P}_{p,n}(y)$ dans le paragraphe précédent. Ensuite, on applique la formule LIBDF d'ordre p en imposant qu'un second polynôme $\psi(t)$ de degré p qui interpole le polynôme $\mathcal{P}_{p,n}(y)$ avec des pas de temps fixes t_{n-1}, $t_{n-1} - h_n$, etc. satisfasse l'EDO suivante en t_n :

$$\begin{aligned} \psi(t_n - ih_n) &= \mathcal{P}_{p,n}(t_n - ih_n), \qquad 1 \leq i \leq p, \\ \psi'(t_n) &= f(t_n, \psi(t_n)), \end{aligned}$$

et en posant $y_n = \psi(t_n)$. La stabilité de cette stratégie est meilleure que la "Fixed Coefficient Stategy" car l'erreur provient d'une seule interpolation, mais moins bonne que celle qui utilise un schéma LIBDF à coefficient variable (on le verra en détail dans la Section 2.3.3). L'utilisation du schéma LIBDF ayant pour but de garantir le maximum de stabilité, on a donc choisi d'utiliser des coefficients variables, stratégie appelée "Variable Coefficient Strategy" qui donne la meilleure stabilité (voir 2.3.3).

Stratégie du coefficient variable (Variable Coefficient Strategy, en abbrégé VC)

Avant d'étudier les questions de stabilité de cette stratégie, il est important de rappeler sa construction. L'idée de cette stratégie est d'utiliser une méthode LIBDF avec des points y_{n-i} en des temps inégalement distants. Pour construire une telle méthode LIBDF, on s'appuie tout d'abord sur les coefficients de la méthode BDF en pas variable. Pour calculer les coefficients du schéma BDF à pas variable on utilise la même méthode que dans le cas pas fixe (à savoir des développements de Taylor), à la différence que les développements de Taylor sont utilisés en pas variable. On obtient ainsi une méthode LIBDF avec des coefficients $\alpha_{i,n}$ et β_n qui dépendent du temps. [CGG90] a donné ainsi la valeur des différents coefficients du schéma BDF à pas variable (qui sont identiques aux coefficients du schéma LIBDF à pas variable). On considère la répartition des points sur la Figure 2.6 :

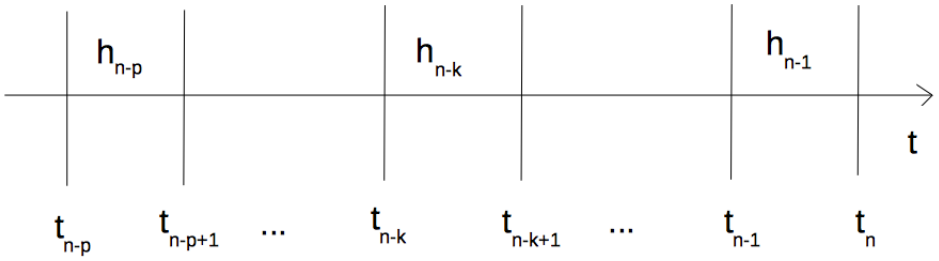


Figure 2.6 – Répartition des temps et des pas de temps

Afin de simplifier les calculs, la forme suivante du schéma BDF est considérée :

$$f(y_n) = \sum_{k=0}^{p} a_k y_{n-k}. \tag{2.119}$$

On a alors les coefficients a_k :

$$a_k = \begin{cases} \displaystyle\sum_{l=1}^{p} \frac{1}{\sum_{j=1}^{l} h_{n-j}} & si \quad k = 0 \\ \displaystyle(-1)^k \frac{\prod_{l=1,\neq k}^{p} \sum_{j=1}^{l} h_{n-j}}{\prod_{l=1}^{k} \sum_{j=l}^{k} h_{n-j} \times \prod_{l=k+1}^{p} \sum_{j=k+1}^{l} h_{n-j}}, & k = 1 \dots p. \end{cases} \tag{2.120}$$

Les coefficients du schéma LIBDF en pas variable (2.19) sont donc :

$$\alpha_{i,n} = -\frac{a_i}{a_0} \qquad i = 1 \dots p \tag{2.121}$$

et

$$\beta_n = \frac{1}{h_n a_0}. \tag{2.122}$$

Zéro-stabilité du schéma LIBDF en pas variable

Dans cette section, on étudie la zéro-stabilité du schéma LIBDF en pas variable pour les trois stratégies. Ces premiers résultats de stabilité vont en particulier permettre de choisir la meilleure stratégie. On a déja vu dans la Section 2.2.3 de ce Chapitre que la zéro-stabilité du schéma LIBDF en pas fixe se démontrait facilement en utilisant la

zéro-stabilité du schéma BDF. Il en est de même pour le pas variable et les critères de zéro-stabilité du schéma LIBDF en pas variable sont les mêmes que les critères du schéma BDF en pas variable.

On a vu dans la section 1.1.3 du Chapitre 1 que le schéma BDF à coefficients constants est zéro-stable pour les ordres 1 à 6. Cependant, dans le cas de coefficients variables ce n'est plus systématiquement le cas. Grigorieff dans [Gri83] donne les conditions pour la zéro-stabilité des schémas multipas, en utilisant en particulier des ratios entres les pas de temps successifs :

$$r_i = h_i/h_{i-1}. \tag{2.123}$$

On peut ainsi réutiliser les valeurs de ratios minimaux et maximaux des pas de temps pour lesquels on a toujours la zéro-stabilité. Dans [CM93], Calvo étudie l'impact du ratio maximal sur la zéro-stabilité du schéma dans un cas bien particulier. Pour cela, Calvo se place dans le cas où les pas de temps sont supposés identiques pour les solutions précédentes et il cherche le ratio maximal pour le calcul de la solution au temps suivant. Calvo donne ainsi pour chaque stratégie et pour chaque ordre du schéma BDF les intervalles limites des ratios, voir Table 2.1. Ces intervalles sont les mêmes pour le schéma LIBDF.

Ordre	INT	FLC	VC
2	$[0, 1.732]$	$[0, 1.732]$	$[0, 2.414]$
3	$[0, 1.406]$	$[0, 1.375]$	$[0, 1.618]$
4	$[0, 1.241]$	$[0, 1.178]$	$[0, 1.280]$
5	$[0, 1.052]$	$[0, 1.109]$	$[0, 1.127]$

TABLE 2.1 – Intervalles de zéro-stabilité de la méthode LIBDF pour les trois stratégies

On remarque que la méthode VC (Méthode à coefficients variables) est la meilleure méthode en terme de zéro-stabilité car elle offre les intervalles de ratios de pas de temps les plus larges. On démontre maintenant que l'on obtient bien les mêmes intervalles de zéro-stabilité pour le schéma LIBDF et pour le schéma BDF. Cette propriété est vraie pour tous les ordres et pour toutes les stratégies, cependant on choisit de le démontrer ici pour la stratégie VC et les ordres 2 et 3.

On rappelle que le schéma LIBDF d'ordre p s'écrit

$$y_{n+1} = \sum_{i=0}^{p-1} \alpha_{i,n} y_{n-i} + \beta_n h_n (f(\mathcal{P}_{p,n}(\boldsymbol{y})) + f'(c)(y_{n+1} - \mathcal{P}_{p,n}(\boldsymbol{y}))). \tag{2.124}$$

On écrit les coefficients $\alpha_{i,n}$ et β_n comme des fonctions du ratio des pas de temps que l'on note $r_i = h_i/h_{i-1}$:

$$\begin{aligned} \alpha_{i,n} &= \alpha_{i,n}(r_n, r_{n-1}, \ldots, r_{n-p+2}), \qquad k = 0, \ldots p-1, \\ \beta_n &= \beta_n(r_n, r_{n-1}, \ldots, r_{n-p+2}). \end{aligned}$$

On construit alors la matrice carrée de taille $p-1$ A_n^* telle que :

$$A_n^* = \begin{pmatrix} \alpha_{0,n}^* & \alpha_{1,n}^* & \ldots & \alpha_{p-2,n}^* \\ 1 & 0 & \ldots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & 1 & 0 \end{pmatrix}$$

où on définit les coefficients $\alpha^*_{i,n}$ en utilisant le polynôme caractéristique du schéma LIBDF :

$$\rho_n(z) = z^p - \sum_{k=0}^{p-1} \alpha_{k,n} z^{p-k-1}. \tag{2.125}$$

A partir de ce polynôme, comme le schéma LIBDF est consistant (par construction des coefficients), on introduit le polynôme réduit :

$$\rho^*_n(z) = \frac{\rho_n(z)}{z-1} = z^{p-1} - \sum_{k=0}^{p-2} \alpha^*_{k,n} z^{p-k-2}. \tag{2.126}$$

On a alors les coefficients de la matrice A^*.

On note N l'indice du dernier point calculé par le schéma et on appelle h le vecteur $h := (h_0, h_1, \ldots, h_N)$. On définit également par H, l'ensemble des vecteurs pour lesquels on a :

$$\max\{h_j | h \in H\} \to 0, \qquad \sup\{h_0 + h_1 + \ldots + h_N | h \in H\} < \infty, \tag{2.127}$$

avec $N = N(h) \to \infty$ pour tout $h \in H$. A partir de l'espace H, on construit l'espace H_0, s'il existe $h^0 > 0$ pour lequel on a :

$$h \in H_0 := \{h \in H | \max h_j < h^0\}. \tag{2.128}$$

A partir de ces coefficients, on donne alors les conditions de zéro-stabilité, initialement donnés par Grigorieff, pour le schéma LIBDF à pas variable d'ordre p

$$\sup\left\{\left\|\prod_{j=k}^{l} A^*_j\right\|; p-1 \le k \le l \le N-1, h \in H\right\} < +\infty, \tag{2.129}$$

$$\sup\left\{\left\|e^T_{p-1} \sum_{j=k}^{l} \left(\prod_{r=k}^{j-1} A^*_r\right)\right\|; p-1 \le k \le l \le N-1, h \in H\right\} < +\infty, \tag{2.130}$$

avec $e^T_{m-1} = (0, \ldots, 0, 1) \in \mathbb{R}^{m-1}$ et $\|\cdot\|$ une norme matricielle convenablement choisie.

Une condition suffisante pour que les deux conditions (2.129) et (2.130) soient satisfaites consiste à trouver une norme telle que :

$$\|A^*_n\| \le q < 1, \qquad n = p-1, \ldots, N-1. \tag{2.131}$$

Le but est donc de construire des normes matricielles satisfaisant la condition (2.131) et tout en permettant le plus grand ratio entre deux pas de temps successifs possibles.

LIBDF Ordre 2

La matrice A_n* est de taille 1 et s'exprime :

$$A^*_n = \alpha^*_{0,n}, \tag{2.132}$$

avec

$$\alpha^*_{0,n} = -\alpha_{1,n} = \frac{h_n}{(h_n + h_{n-1}) h_{n-1} (\frac{1}{h_{n-1}} + \frac{1}{h_n + h_{n-1}})} \tag{2.133}$$

$$= \left(\frac{h_n}{h_{n-1}}\right)^2 \left(\frac{1}{1 + 2\frac{h_n}{h_{n-1}}}\right). \tag{2.134}$$

Un seul ratio de pas de temps est à considérer $r_n = \frac{h_n}{h_{n-1}}$, il vient alors

$$\alpha_{0,n}* = r_n^2 \left(\frac{1}{1+2r_n} \right). \tag{2.135}$$

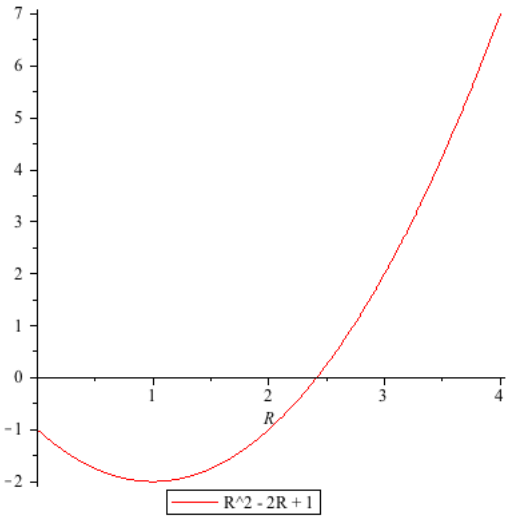
La norme matricielle est la valeur absolue (dimension 1) et on a simplement le critère suivant

$$r_n^2 \left(\frac{1}{1+2r_n} \right) < 1 \tag{2.136}$$

Et on a la condition suivante :

$$r_n^2 - 2r_n - 1 < 0 \tag{2.137}$$

En traçant ce polynôme, on obtient



Le polynôme $r_n^2 - 2r_n - 1 < 0$ a pour racines

$$r_{1,n} = 1 + \sqrt{2}, \tag{2.138}$$

$$r_{2,n} = 1 - \sqrt{2}. \tag{2.139}$$

La seule plage de valeur pour r_n qui vérifie la condition 2.137 est donc

$$r_n \in \left]0, 1+\sqrt{2}\right[. \tag{2.140}$$

LIBDF d'ordre 3

La matrice est la suivante

$$A_n^* = \begin{pmatrix} \alpha_{0,n}^* & \alpha_{1,n}^* \\ 1 & 0 \end{pmatrix},$$

avec $\alpha_{0,n}^* = 1 + \alpha_{1,n}$ et $\alpha_{1,n}^* = -\alpha_{2,n}$.

Une norme possible est :

$$\|A^*\|_\lambda = \|\Lambda^{-1} A^* \Lambda\|_\infty, \tag{2.141}$$

avec $\Lambda = \begin{pmatrix} \lambda & \overline{\lambda} \\ 1 & 1 \end{pmatrix}$.

λ est un nombre complexe avec une partie imaginaire non nulle choisie de telle manière à maximiser les rapports entre deux pas de temps consécutifs.

Après calcul, la valeur optimale est $\lambda_{opt} = 0.4943 + 0.5748I$ et on trace alors en 3D (Figure 2.7) la valeur de la norme $\|A_n^*\|_\lambda$ en fonction des deux ratios de pas de temps que l'on doit considérer pour l'ordre 3 : $r_n = \frac{h_n}{h_{n-1}}$ et $r_{n-1} = \frac{h_{n-1}}{h_{n-2}}$. On ajoute également le plan de cote 1. Tous les points de la surface $\|A_n^*\|_\lambda$ situés en dessous du plan de cote 1 sont donc des points pour lesquels les ratios de pas de temps garantissent la zéro-stabilité.

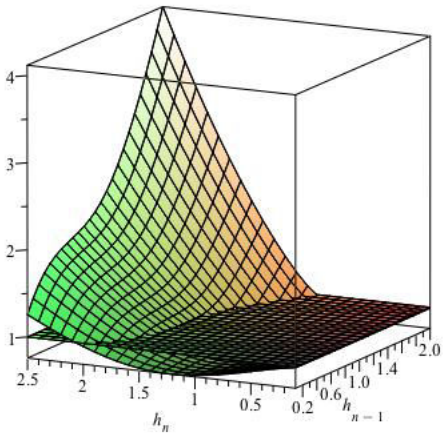


Figure 2.7 – Valeur de $\|A_n^*\|_\lambda$ en fonction des ratios r_n et r_{n-1} et plan de cote 1.

On trace maintenant sur la Figure 2.8 les valeurs de r_n et r_{n-1} qui correspondent exactement à $\|A_n^*\|_\lambda = 1$. La zone en rouge en dessous de la courbe correspond à la zone de zéro-stabilité. Cette courbe donne à r_{n-1} donné la valeur maximale de r_n et réciproquement.

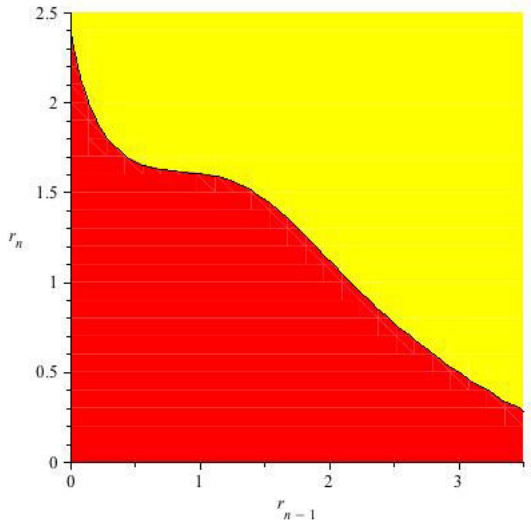


Figure 2.8 – Courbe de r_{n-1} en fonction de r_n qui correspond exactement à $\|A_n^*\|_\lambda = 1$.

On remarque que les ratios de pas de temps, pour l'ordre 3 et l'ordre 2, pour lesquels on a bien zéro-stabilité sont les mêmes que ceux donnés par Calvo dans le tableau 2.1 pour le schéma BDF. Cette propriété est en accord avec le fait que le schéma LIBDF pas fixe et le schéma BDF pas fixe sont équivalents en terme de zéro-stabilité, comme on a pu le voir dans la Section 2.2.3. Ceci est dû au fait que les matrices utilisées dans la démonstration de la zéro-stabilité pour le schéma LIBDF sont les mêmes que les matrices utilisées pour schéma BDF.

On peut alors généraliser le théorème donné par R.D. Grigorieff pour la stratégie VC dans [Gri83] aux schémas LIBDF.

Théorème 2.21. *On appelle p l'ordre du schéma et pour $p = 2, 3, \ldots, 6$, on peut toujours trouver un intervalle $[q_p, Q_p]$ contenant la valeur 1, à l'intérieur duquel le schéma LIBDF est stable si les ratios de pas de temps satisfont :*

$$q_p < \inf\{\frac{h_{j+1}}{h_j}, \quad j = 0, \ldots, N + p - 2, h \in H_0\}, \sup\{\frac{h_{j+1}}{h_j} | \ldots\} < Q_p. \tag{2.142}$$

Les valeurs limites de ratios de pas de temps données ci-dessus permettent d'assurer la condition 2.131, qui n'est qu'une condition suffisante. En pratique, le choix du pas de temps et donc, en particulier du ratio, est dirigé par le contrôle de l'erreur qui permet d'assurer la zéro-stabilité du schéma même si les ratios dépassent localement les conditions ci-dessus.

Absolue stabilité du schéma LIBDF en pas variable

On a vu dans le paragraphe précédent que le schéma LIBDF en pas variable est zéro-stable sous certaines conditions et que la stratégie à coefficients variables (VC) donne la meilleure zéro-stabilité. On a également vu dans la Section 2.2.3 que dans le cas avec pas fixe, les régions de stabilité absolue du schéma LIBDF sont les mêmes que celles du schéma BDF. Il en est de même pour le cas avec pas variable car on applique toujours le schéma numérique à l'équation test de Dahlquist, pour lequel le schéma LIBDF dégénère en schéma BDF. Calvo dans [CM93] donne les régions d'absolue stabilité en fonction du ratio maximal de pas de temps pour les ordres 2, 3 et 4 du schéma BDF et pour les trois stratégies (INT, FLC et VC). Pour chaque ordre, c'est la stratégie à coefficients variables VC qui a la plus grande région d'absolue stabilité (voir Figure 3.1, 3.2 et 3.3 de [CM93]).

Ainsi la stratégie choisie pour le schéma LIBDF pas variable est la stratégie VC, car elle donne la meilleure zéro-stabilité ainsi que la meilleure absolue stabilité. De plus, elle est également la plus économique en calcul pour les systèmes de grande taille car le coût de calcul n'est pas sensible à la taille du problème pour cette stratégie.

2.4 Gestion des évènements

Les systèmes avec évènements sont fréquemment utilisés en ingénierie. Ils mêlent des phénomènes temporels continus et des temps discrets lors des évènements, les systèmes sont alors dits hybrides. Les difficultés de la détection numérique des évènements et des modifications possibles du modèle doivent être résolues dans le contexte du langage Modelica [EM97] [Fri03] et des potentialités du logiciel xMOD [BGCC+10].

[JSC01] a développé plusieurs exemples de systèmes hybrides, typiquement : la balle rebondissante (évènement : le rebond), le thermostat (évènement : la marche et l'arrêt en fonction de la température), la croissance et la division cellulaire (évènement : une cellule se divise en deux cellules). D'autres exemples peuvent être trouvés tels que des modèles de

moteur (évènement : une soupape s'ouvre ou se ferme) ou des modèles de collision [Mos99]. Ces modèles sont non linéaires, raides et de grande taille. Dans ce contexte, des évènements imprévisibles doivent être détectés et le modèle mis à jour. Le processus se répète à chaque évènement donc il est important de déterminer le plus précisément possible le temps de l'évènement et son emplacement sur la trajectoire d'autant plus que le temps de l'évènement et sa position constituent les nouvelles conditions initiales pour l'évolution future du phénomène et affecte définitivement la qualité de la solution. Des benchmarks pour la détection d'évènements et leur gestion sont nécessaires pour apprécier la qualité du solveur.

Durant l'évolution temporelle d'un système dynamique, un évènement peut se produire et modifier le système. Plusieurs types d'évènements sont à prendre en compte [LLK+99]

- les évènements prévisibles : ils se produisent à des temps connus à l'avance et ainsi le pas d'intégration peut être adapté à l'avance,
- les évènements imprévisibles : ils correspondent à une contrainte qui n'est plus satisfaite. A chaque pas de temps, on doit vérifier si la contrainte est respectée ou pas et si elle ne l'est pas, une procédure doit détecter précisément le temps de l'évènement.

Comme exemple d'évènements prévisibles, on cite, dans un circuit électrique le mode ON/OFF d'un interrupteur qui basculerait périodiquement. L'ouverture d'une soupape lorsque la pression est supérieure à un seuil critique est un exemple d'évènement imprévisible.

Dès que le temps de l'évènement est exactement déterminé, le modèle dynamique doit être modifié pour représenter le nouvel état. La résolution numérique du système dynamique produit une trajectoire ; à partir de chaque évènement, une ou plusieurs trajectoires sont modifiées. Le schéma numérique doit détecter l'évènement et faire les modifications du modèle phénoménologique nécessaires.

Il y a également plusieurs types de modifications après un évènement.

- l'apparition d'un évènement peut produire de nouvelles conditions initiales,
- une ou plusieurs équations peuvent être modifiées, mais le nombre d'équations est le même,
- plusieurs équations peuvent être ajoutées/supprimées et le reste des équations modifié.

Le logiciel OpenModelica [Fri03] ne permet pas facilement de gérer les deux derniers types d'évènement.

Les paragraphes suivants présentent la méthode utilisée dans le solveur pour gérer les discontinuités des modèles à évènements. L'application de cette méthode est décrite avec l'exemple de la balle rebondissante dans le Chapitre 4 section 4.1.

2.4.1 Utilisation d'un vecteur de contraintes

Les systèmes dynamiques ont souvent différentes contraintes. Toutes ces contraintes sont les composantes d'un vecteur $\boldsymbol{c}$. Quand une contrainte n'est plus vérifiée, il y a alors un évènement. Il est possible de réécrire toutes les contraintes afin de considérer uniquement un problème de zero-crossing, c'est-à-dire le changement de signe d'une contrainte qui déclenche l'apparition d'un évènement. La détection d'évènements devient alors un problème de détection des zéros des contraintes. Il est également possible de multiplier par -1 la contrainte afin de considérer uniquement les passages par zéro d'une valeur positive vers une valeur négative par exemple.

2.4.2 Expression du système EDO avec des contraintes

Un solveur avec gestion des évènements doit résoudre des systèmes EDOs tels que (2.143) avec le vecteur des contraintes (décrit dans le paragraphe précédent).

$$\begin{cases} t \in I = [t_{deb} = 0, t_{fin}] \subset \mathbb{R}, \\ f : I \times \mathbb{R}^n \to \mathbb{R}^n, \\ y \in \mathbb{R}^n, \\ \dot{y} = f(t, y), \\ x \in \mathbb{R}^k \quad avec \quad k \leq n, \\ c : \mathbb{R}^k \to \mathbb{R}^k, \\ c(x) > 0, \\ y(t = t_{deb}) = y_0. \end{cases} \tag{2.143}$$

2.4.3 Calcul du temps de l'évènement

On a remarqué que lorsqu'une composante du vecteur des contraintes change de signe, un évènement a lieu. Cependant le schéma numérique que l'on utilise (schéma de Runge-Kutta, BDF ou LIBDF) est discret et on a accès uniquement aux solutions du problème aux noeuds et le plus souvent les évènements ont lieu entre ces noeuds. Sur la Figure 2.9, la valeur de la contrainte est tracée en fonction du temps. Le dernier intervalle d'intégration s'étend au delà de zéro qui est le seuil de déclenchement de l'évènement. L'instant auquel la contrainte est égale à zéro est désigné comme temps de l'évènement.

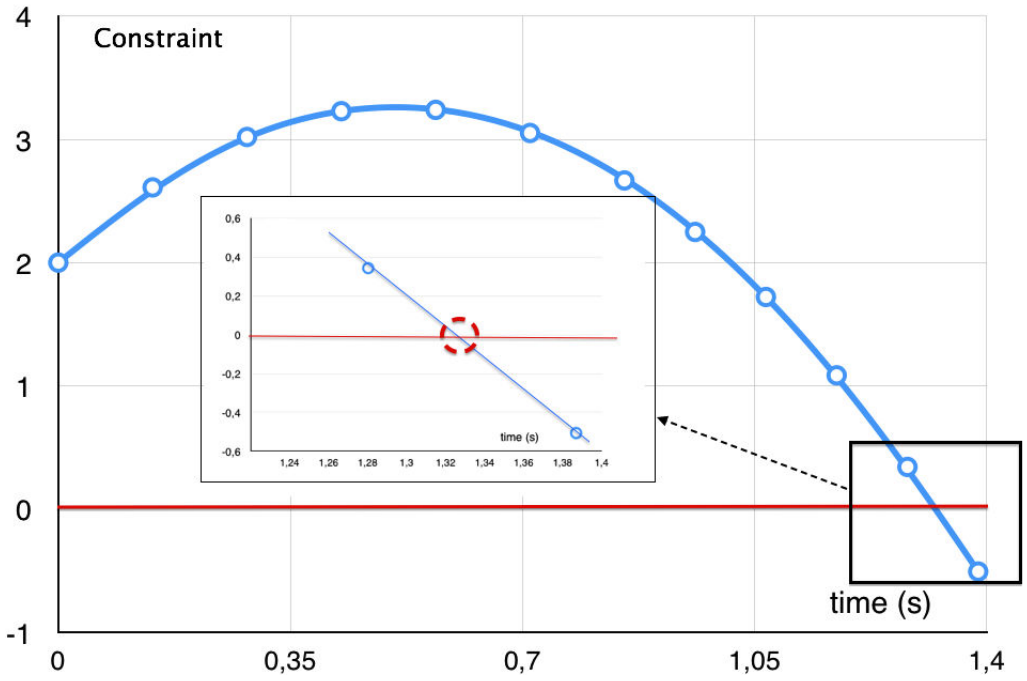


FIGURE 2.9 – Valeur de la contrainte. Zoom sur le dernier intervalle.

Afin de connaître les coordonnées exactes du point d'intersection de la contrainte avec $y = 0$, il est possible d'utiliser une méthode de dichotomie de l'intervalle jusqu'à trouver

le temps t^* auquel la contrainte est nulle ([ZYM08]). Cette méthode est très coûteuse car elle nécessite d'évaluer de nombreuses fois la dérivée $\dot{y} = f(y)$. On présente ici une autre méthode pour calculer t^* qui est beaucoup moins coûteuse.

Au lieu de calculer $\dot{y} = f(y)$, l'idée est d'interpoler un polynôme du second degré passant par les trois dernières valeurs de la contrainte (voir Figure 2.10) ; alors la racine se situant dans l'intervalle approprié est calculée analytiquement. Finalement, on admet que le temps t^* correspond à la racine choisie.

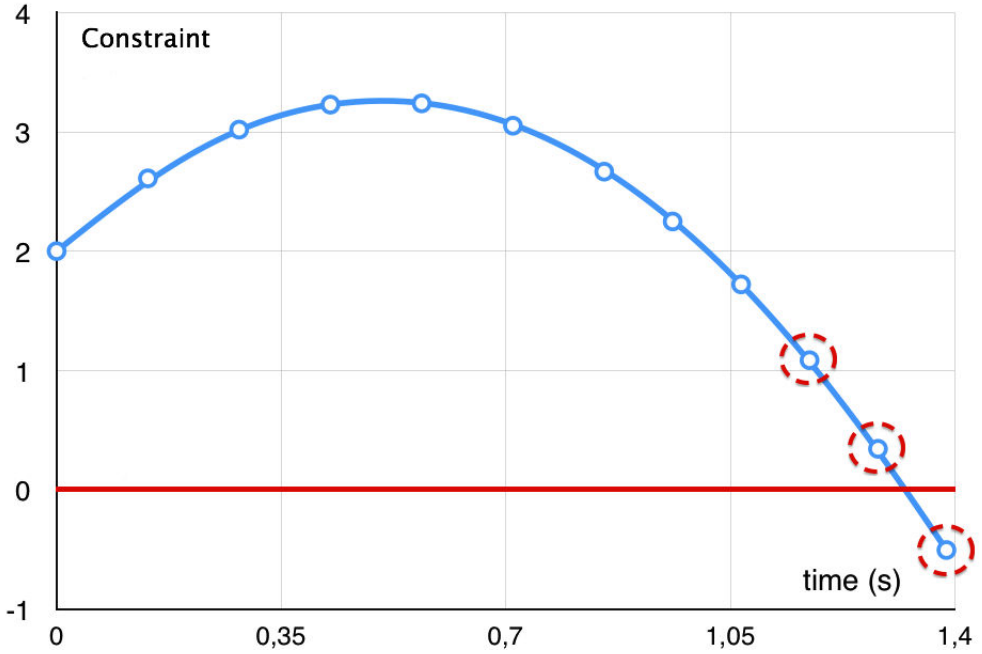


Figure 2.10 – Interpolation à l'aide d'un polynôme en temps du second degré s'appuyant sur les trois points entourés.

En considérant les trois valeurs de la contrainte c_1, c_2, c_3 et leur temps t_1, t_2, t_3 respectivement, l'interpolation polynomiale est :

$$P(t) = \frac{1}{(t_2 - t_3)(t_1 - t_3)(-t_2 + t_1)}((-c_3t_2 + c_1t_2 - t_3c_1 - t_1c_2 + c_3t_1 + t_3c_2)\,t^2 + \left(t_1{}^2c_2 - t_1{}^2c_3 + c_1t_3{}^2 - c_1t_2{}^2 - c_2t_3{}^2 + t_2{}^2c_3\right)t + t_1{}^2c_3t_2 - t_1{}^2t_3c_2 - t_2{}^2c_3t_1 + c_2t_3{}^2t_1 + t_2{}^2t_3c_1 - c_1t_3{}^2t_2).$$

2.4.4 Calcul de la solution en t*

On interpole au moyen d'un polynôme du troisième degré en temps en considérant les deux dernières solutions (t_n, t_{n+1}) et leurs dérivées $(f(y_n), f(y_{n+1}))$.

On évalue ensuite ce polynôme en t* pour obtenir la valeur de la solution $y^* = y(t^*)$ au moment du changement de signe.

2.4.5 Résolution du système après un évènement

Avec l'aide d'une interpolation, on a calculé le temps t^* où le signe de la contrainte change. Cependant, on a besoin de calculer la solution du système à t^*, pour l'utiliser

après comme une condition initiale. Soit y_n le vecteur des variables d'état en t_n. y_n et y_{n+1} sont les solutions calculées juste avant et juste après l'évènement respectivement. Un polynôme du troisième degré en temps est alors interpolé au moyen des deux dernières solutions (y_n, y_{n+1}) et leurs dérivées $(\dot{y}_n, \dot{y}_{n+1})$, voir Figure 2.11. Finalement ce polynôme est évalué en t^* afin de fournir la valeur $y^* \approx y(t^*)$ au moment du changement de signe.

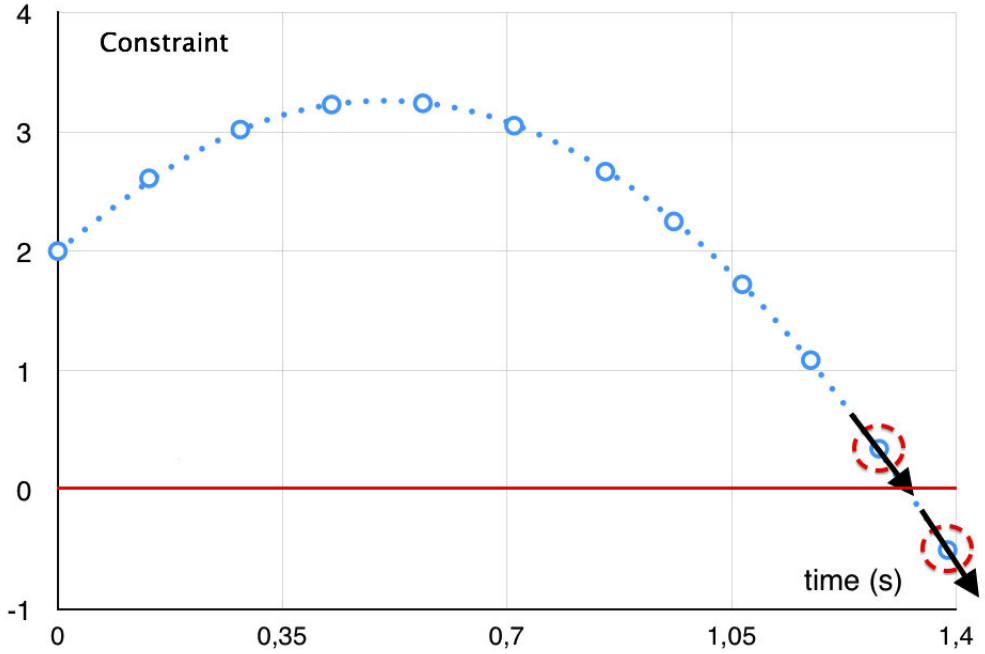


FIGURE 2.11 – Interpolation au moyen d'un polynôme de degré 3 en temps basé sur les deux dernières solutions et leurs dérivées.

2.5 Schéma WFR

Une autre voie d'accélération par l'utilisation de nouveaux schémas numériques est l'utilisation de la méthode WFR (introduite dans la Section 1.1.8 du Chapitre 1). En effet cette méthode permet de diviser le système complet en sous-systèmes qui sont résolus alors en parallèle sur différents processeurs. On propose dans cette section d'étudier les critères de convergence de la méthode WFR. On appliquera essentiellement cette WFR aux méthodes à un pas classiques (Euler explicite et Euler implicite) en pas fixe. On verra dans la Section 2.5.2, la généralisation de la méthode WFR au pas variable.

2.5.1 Convergence de la WFR pour les systèmes linéaires

Un système EDO explicite et linéaire peut s'écrire sous la forme suivante :

$$\begin{cases} t \in I = [t_{start} = 0, t_{end}] \subset \mathbb{R}, \\ X \in \mathbb{R}^n, \\ B \in \mathbb{R}^n, \\ A \in M_n(\mathbb{R}), \\ \dot{X} = AX + B, \\ X(t = t_{start}) = X_0. \end{cases} \tag{2.144}$$

On va alors étudier l'utilisation de la méthode WFR avec deux schémas, le schéma d'Euler explicite et le schéma d'Euler implicite.

Schéma d'Euler explicite

Le schéma d'Euler explicite à pas de temps fixe appliqué au système (2.144) donne :

$$X^i = X^{i-1} + hAX^{i-1} + hB, \tag{2.145}$$

avec X^i la solution approchée au temps $t_i = t_{start} + ih$, et la méthode WFR appliquée à (2.145) donne :

$$X^{i,k+1} = X^{i-1,k+1} + hAX^{i-1,k} + hB, \tag{2.146}$$

avec k la $k - ième$ itération de la méthode WFR.

On suppose que la durée de la simulation avant une nouvelle itération de la WFR est T_w et que pendant cette simulation on a calculé p points.

1er cas : A est diagonalisable. Alors, la matrice de changement de base P de la base initiale à la base des vecteurs propres existe et en multipliant (2.146) par P, on a :

$$PX^{i,k+1} = PX^{i-1,k+1} + hPAP^{-1}PX^{i-1,k} + hPB. \tag{2.147}$$

En notant $Y^{i,k} = PX^{i,k+1}$ et $D = PAP^{-1}$, la matrice diagonale (2.147) est réécrite :

$$Y^{i,k+1} = Y^{i-1,k+1} + hDY^{i-1,k} + hPB. \tag{2.148}$$

Le vecteur Y a les composantes suivantes $Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$. En notant $(\lambda_1, \ldots, \lambda_n)$ les valeurs propres, on a

$$\begin{aligned} y_1^{i,k+1} &= y_1^{i-1,k+1} + h\lambda_1 y_1^{i-1,k}, \\ &\vdots \\ y_n^{i,k+1} &= y_n^{i-1,k+1} + h\lambda_n y_n^{i-1,k}. \end{aligned}$$

Pour une composante j donnée, en faisant varier $1 \leq i \leq p$, on obtient le système linéaire suivant :

$$\begin{pmatrix} y_j^1 \\ \vdots \\ y_j^p \end{pmatrix}^{,k+1} = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} y_j^1 \\ \vdots \\ y_j^p \end{pmatrix}^{,k+1} + \lambda h \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} y_j^1 \\ \vdots \\ y_j^p \end{pmatrix}^{,k} + \begin{pmatrix} y_j^0 \\ \vdots \\ 0 \end{pmatrix} \quad j = 1 \ldots n \tag{2.149}$$

On nomme le vecteur $\begin{pmatrix} y_j^1 \\ \vdots \\ y_j^p \end{pmatrix}$, le vecteur Y_j et on a alors l'équation suivante pour chaque composante du vecteur d'état :

$$Y_j^{k+1} = CY_j^{k+1} + \lambda hCY_j^k + F, \qquad j = 1, \ldots, n, \tag{2.150}$$

avec $C = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix}$ et $F = \begin{pmatrix} y_j^0 \\ \vdots \\ 0 \end{pmatrix} + hPB$. On fait passer les termes correspondants à l'itération $k+1$ de la WFR à gauche :

$$(I - C)Y_j^{k+1} = \lambda h C Y_j^k + F, \qquad j = 1, \dots, n, \tag{2.151}$$

avec $(I - C) = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ -1 & 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 1 \end{pmatrix}$ qui est inversible et qui a pour inverse

$$(I - C)^{-1} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 1 & \dots & 1 & 1 & 1 \end{pmatrix}$$

On obtient alors :

$$Y_j^{k+1} = \lambda h (I - C)^{-1} C Y_j^k + (I - C)^{-1} F, \tag{2.152}$$

avec

$$(I - C)^{-1} C = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 1 & \dots & 1 & 1 & 0 \end{pmatrix}.$$

Une condition suffisante pour que la suite (Y_j^n) converge est que le rayon spectral de la matrice $\lambda h(I - C)^{-1}$ soit strictement plus petit que 1. On utilise ici la norme infinie pour des raisons pratiques, en particulier car la vitesse de convergence est connue, et la norme infinie de la matrice $(I - C)^{-1}C$ est $p - 1$ (p est la taille de la fenêtre de simulation). Donc la suite $(Y_j)^n$ converge si et seulement si $\lambda h(p-1)$ est plus petit que 1. Maintenant que l'on sait que la méthode converge, on doit montrer qu'elle converge vers la bonne solution.

On considère (2.149) sans aucune méthode WFR :

$$Y_j = CY_j + \lambda h C Y_j + F. \tag{2.153}$$

(2.153) correspond au schéma d'Euler explicite appliqué au vecteur Y_j. En soustrayant (2.153) et (2.146), on obtient l'erreur de la méthode WFR par rapport au schéma d'Euler explicite à la $k - ième$ itération E_j^k :

$$E_j^{k+1} = CE_j^{k+1} + \lambda h C E_j^k,$$
$$E_j^{k+1} = (I - C)^{-1} C \lambda h E_j^k.$$

De manière identique, E_j^{k+1} converge vers 0 si et seulement si $\lambda h(p-1)$ est strictement plus petit que 1.

On remarque que la convergence est liée à la taille de la fenêtre de simulation. En effet, plus la taille de la fenêtre de simulation est élevée plus les valeurs "gelées" des variables sont gelées longtemps et ainsi il sera plus long de faire converger l'algorithme WFR. On voit ici que la WFR s'applique très mal aux systèmes raides car dans le cas des systèmes raides la valeur de λ est très elevée et il faut alors soit une taille de fenêtre de simulation très petite, soit un pas de temps très petit.

2ème cas : A n'est pas diagonalisable mais trigonalisable dans $\mathbb{C}$

Comme tout matrice est trigonalisable dans $\mathbb{C}$ ce cas regroupe l'ensemble des cas. Alors, la matrice de changement de base P de la base initiale vers la base de la matrice triangulaire inférieure existe et en multipliant (2.146) par P, cela donne :

$$PX^{i,k+1} = PX^{i-1,k+1} + hPAP^{-1}PX^{i-1,k} + hPB. \tag{2.154}$$

En notant $Y^{i,k} = PX^{i,k+1}$ et $T = PAP^{-1}$, la matrice triangulaire inférieure (2.154) est réécrite :

$$Y^{i,k+1} = Y^{i-1,k+1} + hTY^{i-1,k} + hPB. \tag{2.155}$$

Le vecteur Y a les composantes suivantes $Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$.

En notant $(t_{i,j})$ les coefficients de T :

$$\begin{aligned} y_1^{i,k+1} &= y_1^{i-1,k+1} + h\sum_{m=0}^{n} t_{1m}y_m^{i-1,k}, \\ &\vdots \\ y_n^{i,k+1} &= y_n^{i-1,k+1} + h\sum_{m=0}^{n} t_{nm}y_m^{i-1,k}, \end{aligned}$$

cela conduit à :

$$\hat{Y}^{k+1} = \hat{C}\hat{Y}^{k+1} + h\hat{T}\hat{C}\hat{Y}^{k} + \hat{F}, \tag{2.156}$$

avec : $\hat{Y}^{k+1} = \begin{pmatrix} y_1^{1,k+1} \\ \vdots \\ y_1^{p,k+1} \\ \vdots \\ y_n^{1,k+1} \\ \vdots \\ y_n^{p,k+1} \end{pmatrix}$, $\hat{C} = \begin{pmatrix} M & & 0 \\ & \ddots & \\ 0 & & M \end{pmatrix}$ (avec $M = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix}$),

$\hat{T} = \begin{pmatrix} \tilde{T}_{11} & & 0 \\ \vdots & \ddots & \\ \tilde{T}_{n1} & \dots & \tilde{T}_{nn} \end{pmatrix}$ avec $\tilde{T}_{ij} = \begin{pmatrix} t_{ij} & & 0 \\ & \ddots & \\ 0 & & t_{ij} \end{pmatrix}$,

$$\hat{F} = \begin{pmatrix} y_1^0 \\ 0 \\ \vdots \\ y_2^0 \\ 0 \\ \vdots \\ y_n^0 \end{pmatrix} + hPB,$$

et on a :

$$\hat{Y}^{k+1} = h(I - \hat{C})^{-1}\hat{T}\hat{C}\hat{Y}^k + (I - \hat{C})^{-1}\hat{F}. \tag{2.157}$$

On en déduit que la méthode WFR pour les matrices non diagonalisables converge si la norme infinie de $h(I - \hat{C})^{-1}\hat{T}\hat{C}$ est plus petite que 1.

Maintenant que l'on sait que la méthode converge, on doit montrer qu'elle converge vers la bonne solution.

On considère (2.155) sans aucune méthode WFR :

$$\hat{Y} = \hat{C}\hat{Y} + h\hat{T}\hat{C}\hat{Y} + \hat{F}. \tag{2.158}$$

(2.158) correspond au schéma d'Euler explicite appliqué au vecteur $\hat{Y}$. En soustrayant (2.158) et (2.155), on obtient l'erreur de la méthode WFR par rapport au schéma d'Euler explicite à la $k - ième$ itération $\hat{E}^k$:

$$\hat{E}^{k+1} = \hat{C}\hat{E}^{k+1} + h\hat{T}\hat{C}\hat{E}^k,$$
$$\hat{E}^{k+1} = (I - \hat{C})^{-1}\hat{T}\hat{C}h\hat{E}^k.$$

De la même manière que précédemment, $\hat{E}^{k+1}$ converge vers 0 si la norme infinie de $(I - \hat{C})^{-1}\hat{T}\hat{C}$ est strictement plus petite que que 1.

Schéma d'Euler implicite

Le schéma d'Euler implicite à pas fixe appliqué à (2.144) donne :

$$X^i = X^{i-1} + hAX^i + hB, \tag{2.159}$$

avec X^i la solution approchée au temps $t_i = t_{start} + ih$.

La méthode WFR appliquée à (2.159) donne :

$$X^{i,k+1} = X^{i-1,k+1} + hAX^{i,k} + hB, \tag{2.160}$$

avec k la $k - ième$ itération de la méthode WFR.

On suppose que la durée de la simulation avant une nouvelle itération de la WFR est T_w et que pendant cette simulation on a calculé p points.

1er cas : A est diagonalisable Alors, la matrice de changement de base P de la base initiale à la base des vecteurs propres existe et en multipliant (2.160) par P, on a :

$$PX^{i,k+1} = PX^{i-1,k+1} + hPAP^{-1}PX^{i,k} + hPB. \tag{2.161}$$

En notant $Y^{i,k} = PX^{i,k+1}$ et $D = PAP^{-1}$ la matrice diagonale (2.161) est réécrite :

$$Y^{i,k+1} = Y^{i-1,k+1} + hDY^{i,k} + hPB. \tag{2.162}$$

Le vecteur Y a les composantes suivantes $Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$. En notant $(\lambda_1, \ldots, \lambda_n)$ les valeurs propres, on a :

$$\begin{aligned} y_1^{i,k+1} &= y_1^{i-1,k+1} + h\lambda_1 y_1^{i,k}, \\ &\vdots \\ y_n^{i,k+1} &= y_n^{i-1,k+1} + h\lambda_n y_n^{i,k}. \end{aligned}$$

Toutes les composantes du vecteur Y sont indépendantes les unes des autres et pour chaque composante le vecteur de taille p suivant est construit :

$$\begin{pmatrix} y_j^1 \\ \vdots \\ y_j^p \end{pmatrix}^{,k+1} = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} y_j^1 \\ \vdots \\ y_j^p \end{pmatrix}^{,k+1} + \lambda h \begin{pmatrix} y_j^1 \\ \vdots \\ y_j^p \end{pmatrix}^{,k} + \begin{pmatrix} y_j^p \\ \vdots \\ 0 \end{pmatrix}, \qquad j =, 1, \ldots, n. \tag{2.163}$$

On nomme le vecteur $\begin{pmatrix} y_j^1 \\ \vdots \\ y_j^p \end{pmatrix}$, le vecteur Y_j et on a alors l'équation suivante pour chaque composante du vecteur d'état :

$$Y_j^{k+1} = CY_j^{k+1} + \lambda h Y_j^k + F, \qquad j = 1, \ldots, n, \tag{2.164}$$

avec $C = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 & 0 \end{pmatrix}$ et $F = \begin{pmatrix} y_j^0 \\ \vdots \\ 0 \end{pmatrix} + hPB$. On fait passer les termes correspondant à l'itération $k+1$ de la WFR à gauche :

$$(I - C)Y_j^{k+1} = \lambda h Y_j^k + F, \qquad i = j, \ldots, n, \tag{2.165}$$

avec $(I - C) = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ -1 & 1 & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & -1 & 1 \end{pmatrix}$ qui est inversible et qui a pour inverse

$$(I - C)^{-1} = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & 1 & 0 & \ldots & 0 \\ 1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 1 & \ldots & 1 & 1 & 1 \end{pmatrix}.$$

On obtient alors

$$Y_j^{k+1} = \lambda h (I - C)^{-1} Y_j^k + (I - C)^{-1} F. \tag{2.166}$$

Comme précédemment, on utilise ici la norme infinie pour des raisons pratiques et la norme infinie de la matrice $(I-C)^{-1}$ est p (p est la taille de la fenêtre de simulation).Donc la suite $(Y_j)^n$ converge si et seulement si λhp est plus petit que 1. Maintenant que l'on sait que la méthode converge, on doit montrer qu'elle converge vers la bonne solution.

On considère (2.163) sans aucune méthode WFR :

$$Y_j = CY_j + \lambda hY_j + F. \tag{2.167}$$

(2.167) correspond au schéma d'Euler implicite appliqué au vecteur Y_j. En soustrayant (2.167) et (2.163), on obtient l'erreur de la méthode WFR à la $k - ième$ itération E_j^k :

$$E_j^{k+1} = CE_j^{k+1} + \lambda hE_j^k,$$
$$E_j^{k+1} = (I-C)^{-1}\lambda hE_j^k.$$

De manière identique, E_j^{k+1} converge vers 0 si et seulement si λhp est strictement plus petit que 1.

2ème cas : A n'est pas diagonalisable mais trigonalisable dans $\mathbb{C}$

Comme tout matrice est trigonalisable dans $\mathbb{C}$ ce cas regroupe l'ensemble des cas. Alors, la matrice de changement de base P de la base initiale vers la base de la matrice triangulaire inférieure existe et est multipliée (2.160) par P, cela donne :

$$PX^{i,k+1} = PX^{i-1,k+1} + hPAP^{-1}PX^{i,k} + hPB. \tag{2.168}$$

En notant $Y^{i,k} = PX^{i,k+1}$ et $T = PAP^{-1}$, la matrice triangulaire inférieure (2.168) est réécrite :

$$Y^{i,k+1} = Y^{i-1,k+1} + hTY^{i,k} + hPB \tag{2.169}$$

Le vecteur Y a les composantes suivantes $Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$.

En notant $(t_{i,j})$ les coefficients de T :

$$y_1^{i,k+1} = y_1^{i-1,k+1} + h\sum_{m=0}^{n} t_{1m}y_m^{i,k},$$
$$\vdots$$
$$y_n^{i,k+1} = y_n^{i-1,k+1} + h\sum_{m=0}^{n} t_{nm}y_m^{i,k},$$

cela conduit à :

$$\hat{Y}^{k+1} = \hat{C}\hat{Y}^{k+1} + h\hat{T}\hat{Y}^k + \hat{F}, \tag{2.170}$$

avec : $\hat{Y}^{k+1} = \begin{pmatrix} y_1^{1,k+1} \\ \vdots \\ y_1^{p,k+1} \\ \vdots \\ y_n^{1,k+1} \\ \vdots \\ y_n^{p,k+1} \end{pmatrix}$, $\hat{C} = \begin{pmatrix} M & & 0 \\ & \ddots & \\ 0 & & M \end{pmatrix}$ (avec $M = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix}$),

$$\hat{T} = \begin{pmatrix} \tilde{T}_{11} & & 0 \\ \vdots & \ddots & \\ \tilde{T}_{n1} & \dots & \tilde{T}_{nn} \end{pmatrix} \text{ avec } \tilde{T}_{ij} = \begin{pmatrix} t_{ij} & & 0 \\ & \ddots & \\ 0 & & t_{ij} \end{pmatrix},$$

$$\hat{F} = \begin{pmatrix} y_1^0 \\ 0 \\ \vdots \\ y_2^0 \\ 0 \\ \vdots \\ y_n^0 \end{pmatrix} + hPB,$$

et on a :

$$\hat{Y}^{k+1} = h(I - \hat{C})^{-1}\hat{T}\hat{Y}^k + (I - \hat{C})^{-1}\hat{F}. \tag{2.171}$$

On en déduit que la méthode WFR pour les matrices non diagonalisables converge si la norme infinie de $h(I - \hat{C})^{-1}\hat{T}$ est plus petite que 1.

Maintenant que l'on sait que la méthode converge, on doit montrer qu'elle converge vers la bonne solution.

On considère (2.169) sans aucune méthode WFR :

$$\hat{Y} = \hat{C}\hat{Y} + \hat{T}h\hat{Y} + \hat{F}. \tag{2.172}$$

(2.172) correspond au schéma d'Euler implicite appliqué au vecteur $\hat{Y}$. En soustrayant (2.172) et (2.169), on obtient l'erreur de la méthode WFR à la $k - ième$ itération $\hat{E}^k$:

$$\hat{E}^{k+1} = \hat{C}\hat{E}^{k+1} + \hat{T}h\hat{E}^k,$$
$$\hat{E}^{k+1} = (I - \hat{C})^{-1}\hat{T}h\hat{E}^k.$$

De la même manière que précedemment, $\hat{E}^{k+1}$ converge vers 0 si la norme infinie de $\hat{T}h(I - \hat{C})^{-1}$ est strictement plus petite que que 1.

2.5.2 Application de la méthode WFR en pas variable

La méthode WFR s'applique très mal dans le cas du pas variable. En effet à la fin d'une wave les solutions de chaque sous-systèmes sont échangées, cependant si le pas de temps est variable, les solutions qui seront échangées n'auront pas été calculées aux mêmes temps. Ainsi, lors de l'itération suivante, on ne pourra pas utiliser ces solutions en l'état. Il faudra alors faire des interpolations. Cependant il est possible qu'un sous-système soit beaucoup plus raide qu'un autre et ainsi utilise des pas de temps beaucoup plus petits. L'interpolation avec les solutions des autres sous-systèmes sera alors très mauvaise. Ainsi l'utilisation de la WFR est limitée au cas du pas fixe et empèche d'avoir un contrôle de l'erreur lors de la simulation car on ne peut pas adapter le pas de temps suivant l'erreur commise. De plus, si on veut s'assurer que la solution converge il est nécessaire d'utiliser un pas de temps fixe assez petit ce qui augmente très fortement le temps de calcul.

2.5.3 Amélioration de la Waveform Relaxation

L'efficacité de la méthode, c'est-à-dire la vitesse de convergence, dépend de la qualité du partitionnement du système et fonctionne mieux avec des sous-systèmes faiblement couplés. Dans [HC90], trois types de couplage sont comparés afin de montrer comment

ils affectent la convergence de la solution. De plus, il est montré dans [BDP96] ainsi que dans la Section 2.5.1, que plus la taille de la fenêtre de simulation est grande, plus lente est la convergence. Dans ce contexte, différents paramètres sont étudiés afin d'améliorer le nombre d'itérations :

Contrôle de la tolérance du solveur en fonction de l'itération de la WFR

En effet, lors des premières itérations de la WFR, les résultats ne sont pas précis en raison du manque de données provenant des autres sous-systèmes. L'idée est alors d'augmenter la tolérance du solveur uniquement pour les premières itérations, puis progressivement de la diminuer.

Initialisation avec une perturbation faible de l'état d'équilibre

En effet, avec une initialisation proche de la solution, peu d'itérations sont nécessaires pour que la WFR converge.

Utilisation de fenêtres de simulation

On a vu dans la Section 2.5.1 qu'on utilisait des fenêtres de simulation dont la taille T_w était plus petite que la durée totale de simulation t_{end}. Cela permet de faciliter la convergence et de diminuer le nombre d'itérations ; en effet la Section 2.5.1 a permis de démontrer que la convergence de la WFR est directement liée à la taille de cette fenêtre de simulation. Ainsi, la taille de la fenêtre suivante peut être redimensionnée au cours de l'intégration par rapport au nombre d'itérations nécessaire pour la fenêtre courante. Si le nombre d'itérations est trop élevé, on diminue la taille de la prochaine fenêtre de simulation ou au contraire si le nombre d'itérations est faible, on peut augmenter la taille de la prochaine fenêtre. De plus, plus la taille de la fenêtre simulation est grande moins il y aura besoin de faire de communications entre les différents processeurs qui travaillent sur les sous-systèmes. En particulier, pour les problèmes peu raides, on peut choisir une très grande taille de fenêtre et ainsi minimiser le nombre d'échange de données entre les processeurs.

Chapitre 3

Accélération par une parallélisation efficace

Le Chapitre précédent a développé les nouveaux outils numériques mis en place pour l'implémentation d'un solveur EDO qui permettent de résoudre rapidement des problèmes fortement non linéaires et raides. Cette problématique de rapidité d'exécution est d'autant plus importante que la taille du problème à résoudre est importante. Ainsi le schéma LIBDF décrit dans la Section 2.2 du Chapitre 2 minimise les coûts de calcul tout en garantissant une grande stabilité. Dans le cadre des problèmes de grande taille il existe un moyen informatique de résoudre plus rapidement les systèmes EDO, il s'agit de la parallélisation du solveur. Ce Chapitre va détailler les deux grands types de parallélisation effectués sur le solveur EDO développé lors de la thèse. Afin d'illustrer l'efficacité de ces parallélisations sur le solveur EDO, nous nous sommes également attachés à proposer une méthode de construction de benchmarks permettant de valider une stratégie de parallélisation, et de mettre en avant tel ou tel choix de parallélisation. On a choisi de séparer la méthode WFR de ce Chapitre pour la placer dans le Chapitre précédent.

Sommaire

3.1 Profiling du solveur

Avant de commencer la parallélisation du solveur EDO, il est important d'étudier le solveur et en particulier de déterminer les parties du code qui sont les plus coûteuses afin de concentrer les efforts de parallélisation sur ces parties.

On rappelle la définition de speedup :

Définition 3.1. Le speedup correspond au gain de temps lors de l'exécution d'un programme en utilisant plusieurs processeurs ou coeurs par rapport au temps d'exécution du programme séquentiel :

$$S(n) = \frac{T(1)}{T(p)}, \tag{3.1}$$

avec $T(1)$ le temps d'exécution du programme séquentiel et $T(n)$ le temps d'exécution du programme en utilisant n unités de calcul.

De plus, il est également primordial d'étudier le pourcentage de code qu'il est possible de paralléliser afin de prévoir les speedups maximaux que l'on pourra attendre selon la loi d'Amdahl. En effet la loi d'Amdahl est utilisée afin de déterminer l'amélioration maximale prévue d'un système quand seulement une partie de ce système est améliorée. Bien que cette loi soit générale, elle est très souvent appliquée dans le cas de calcul parallèle afin de prédire le speedup maximal en utilisant plusieurs processeurs ou plusieurs coeurs.

Cette loi exprime le fait que l'accélération d'un programme utilisant plusieurs processeurs/coeurs est limitée par le temps nécessaire pour la partie séquentielle du programme.

Ainsi, si on nomme $n \in \mathbb{N}$ le nombre de tâches d'exécution, $B \in [0, 1]$, la proportion de l'algorithme qui est uniquement séquentielle, le temps $T(n)$ qu'un algorithme prend pour s'exécuter quand il est exécuté sur n tâches correspond à :

$$T(n) = T(1)\left(B + \frac{1}{n}(1 - B)\right). \tag{3.2}$$

Ainsi, le speedup théorique $S(n)$ qui peut être obtenu en exécutant un tel algorithme d'un système capable d'exécuter n tâches est :

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1)(B + \frac{1}{n}(1 - B))} = \frac{1}{B + \frac{1}{n}(1 - B)}. \tag{3.3}$$

Ainsi si par exemple on ne parallélise que 12% du code, on n'aura au maximum que 1.136 de speedup. Il est plus habituel de remplacer la notation B par la proportion du programme que l'on peut paralléliser P. On a alors :

$$S(n) = \frac{1}{(1 - P) + \frac{P}{n}}. \tag{3.4}$$

Ainsi si le nombre de tâches n tend vers l'infini le speedup maximal est $1/(1-P)$. Par exemple si P est de 90%, alors le programme peut être accéléré par un facteur 10 seulement, peu importe le nombre de tâches utilisées. On trace sur la figure suivante (Figure 3.1), le speedup maximal attendu en fonction du nombre de processeurs pour quatre proportions de parallélisation 50%, 75%, 90% et 95%.

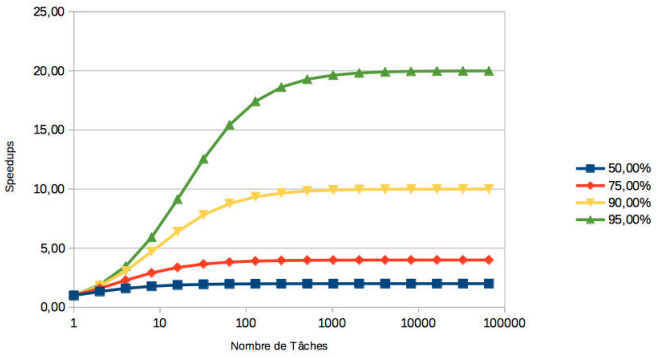


Figure 3.1 – Speedup en fonction du nombre de processeurs

.

Regardons maintenant les sections particulièrement coûteuses du solveur implémenté dans la thèse qui utilise en particulier le nouveau schéma LIBDF. On utilise l'outil de profiling du logiciel Xcode afin de connaître le coût de chaque partie. On obtient alors de cette analyse, par ordre décroissant, pour un système à 1000 équations :

- évaluation de la dérivée : 30.9%,
- résolution de la partie implicite du schéma LIBDF : 23.6%,
- initialisation des différentes variables : 9.5%,
- stockage de la solution : 8.8%,
- calcul de la partie explicite du schéma LIBDF : 8.6%,
- Calcul de l'interpolation de Lagrange 7.9%.

Comme on pouvait s'y attendre c'est l'évaluation de la dérivée qui est la plus coûteuse, suivi par la résolution de la partie implicite du schéma LIBDF (l'inversion du système linéaire).

On verra dans la Section 3.3 et la Section 3.4 comment paralléliser ces différentes sections (si cela est possible).

3.2 Méthodes de construction de benchmarks

Les récentes années ont vu le développement de méthodes parallèles efficaces pour la résolution numérique des systèmes EDOs. En effet le besoin de solveurs plus rapides a conduit à utiliser des ordinateurs en parallèle et des systèmes distribués. Le principal défi est de tirer parti de cet énorme potentiel de puissance de calcul. Afin d'être efficace, un tel solveur doit s'appuyer sur des algorithmes qui sont bien adaptés à ces nouvelles architectures. Pour toutes ces raisons, on propose un benchmark qui met en avant l'influence des stratégies de parallélisation de solveurs d'EDO de grande taille sur le speedup. Le choix du solveur est ici mis sous silence comme sa problématique a déjà été détaillée dans le Chapitre 2.

"Le Benchmarking des méthodes EDO est une longue tradition," déclare Nowak [NG97] qui produit un réel effort pour décrire un nouveau cadre de tests pour les solveurs EDO.

Dans [SVVL+97], Sandu définit sept problèmes tests dans le contexte de la chimie atmosphérique dans le but de comparer des solveurs tels que QSSA, Lsode, Chemeq ... Liu dans [LFF10] compare quatre solveurs utilisant un modèle non raide, raide et hybride. Ces références ne traitent que des problèmes assez petits incluant certes des difficultés spécifiques mais ne sont pas concentrées sur des problèmes de grande taille qui sont soumis à des exigences de parallélisation.
Le benchmark proposé ici donne la possibilité d'ajuster la taille du problème exprimé en termes de nombre d'éléments non nuls de la matrice résolvante, permettant de comparer les temps de calcul pour des problèmes de taille significativement différente. Cela ne peut pas être traité avec un problème EDO industriel courant. Pour la construction de ce benchmark, on commence avec un modèle de système linéaire représenté par une matrice n-diagonale, avec une solution exacte disponible quelle que soit la taille de la matrice. On procède alors en effectuant la rotation de deux vecteurs de base de $\mathbb{R}^n$ correspondant à un angle de rotation choisi. Ce changement de base introduit itérativement une augmentation du nombre de termes non-nuls et ainsi une nouvelle complexité dans le calcul du problème même si cette complexité est artificielle. Par ce moyen, on formule de nombreux modèles linéaires avec différents couplages dont les solutions sont connues et toutes identiques. Le nombre d'éléments non nuls de la matrice permet de distinguer les performances de différents schémas numériques, les méthodes de parallélisation comme un premier pas pour une certification des futurs solveurs.

La famille de modèles incluant plus ou moins de termes non nuls et augmentant en complexité peut être vue comme une approche différente de Design Structure Matrix. Tyson Browning dans son livre [EB12] propose une méthode pour obtenir une matrice bloc diagonale d'une matrice creuse. A l'inverse de lui on propose ici d'obtenir une matrice progressivement dense à partir d'une matrice diagonale.

De plus, cette méthode marche sur une matrice fonctionnelle et pas sur des connexions de graphe.

On a implémenté cette approche sur un modèle d'un phénomène physique linéaire 1D de conduction de la chaleur dans une tige avec des sections successives ayant d'identiques (problèmes linéaires) ou différentes (problèmes non linéaires) conductivités thermiques. Les résultats de cette application sont présentés dans le Chapitre 4 Section 4.4.

3.2.1 Besoin de la matrice d'un problème de grande taille

Le problème général avec $t \in [t_{start}, t_{end}] \subset \mathbb{R}$ et $(X, \dot{X}) \in \mathbb{R}^n \times \mathbb{R}^n$ est défini comme :

$$\begin{cases} \dot{X}(t) = f(X(t), t), \\ X(t=0) = X_0. \end{cases} \tag{3.5}$$

La formulation (3.5) est la plus courante et comme notre but est de travailler sur des matrices pour ajouter du couplage, la formulation suivante est utilisée :

$$\begin{cases} \dot{X}(t) = A\ X(t) + B, \\ X(t=0) = X_0. \end{cases} \tag{3.6}$$

Le vecteur $B \in \mathbb{R}^n$ et la matrice résolvante $A \in M_n(\mathbb{R})$ ne sont pas toujours explicitement connus. On propose dans ce paragraphe différentes méthodes pour obtenir A et B de grande taille dans le cas de systèmes linéaires.

Premier cas : La dérivée spatiale conditionne la taille

Le modèle (3.5) peut être le résultat de la discrétisation spatiale d'une EDP par éléments finis ou volumes finis [TZL], [Sch91]. La taille de ce problème est alors ajustable en fonction du nombre de noeuds utilisés pour la discrétisation et il est possible d'augmenter la taille du système autant que voulu. La discrétisation fournit également les coefficients de la matrice A et du vecteur B.

Deuxième cas : Par lecture directe de f

Si (3.5) ne contient pas de dérivée spatiale et est déjà de grande taille, les coefficients de la matrice A et du vecteur B sont déterminés en lisant directement les équations de la fonction f.

Troisième cas : Le modèle provient d'un outil de simulation et est décrit dans un langage équationnel

L'outil de simulation peut être OpenModelica et le langage de modélisation Modelica. Modelica est un langage orienté objet, déclaratif et multidomaines pour la modélisation orientée composant de systèmes complexes, par exemple, les systèmes contenant des composants mécaniques, électriques, hydrauliques, thermiques... (voir [Fri03] et la Section 1.3 du Chapitre 1). Modelica fournit différents exemples de modèles complexes mais généralement faiblement couplés. Il n'est pas évident de trouver les coefficients de la matrice A et du vecteur B correspondant au problème entier à moins que le problème ne soit très simple et ne contienne que peu d'équations. Nous proposons ici une méthode pour déterminer ces coefficients pour un modèle de grande taille issu de Modelica.

Une caractéristique de Modelica est qu'il est capable d'exporter des modèles entiers via un format standard appelé FMU (Functional Mockup Unit, voir la Section 1.3 du Chapitre 1). Ce FMU contient une fonction appelée getderivatives qui correspond à la fonction f de (3.5).

Identifions le vecteur B et appliquons le vecteur nul à la fonction f ; alors, il vient directement $B = f(0)$.

Pour identifier les coefficients de A, on applique chaque vecteur de la base canonique à la fonction f, typiquement $e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$, $e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$, $\ldots$ $e_n = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$.

Avec e_1 on obtient un vecteur qui est la première colonne de la matrice A. En itérant le processus avec tous les vecteurs de la base canonique, on obtient toutes les colonnes de A qui sont stockées dans une structure de type CSR (voir Section 4.4.5 du Chapitre 4).

Une telle identification est coûteuse en terme de calcul ; de plus, ce processus d'identification est pertinent si la fonction f est linéaire. Si f n'est pas linéaire, l'identification devient plus difficile et est seulement possible dans des cas particuliers. Même si on dispose de tels modèles linéaires provenant d'applications industrielles, deux propriétés indispensables ne sont généralement pas vérifiées : premièrement, nous ne connaissons pas la solution exacte du problème industriel, principalement s'il est de très grande taille ; deuxièmement, cela n'a pas de sens de résoudre séparément certaines parties du modèle,

bien que chaque partie soit de taille plus petite et soit plus facilement manipulable.
C'est pourquoi, les modèles industriels de grande taille ne sont pas adaptés pour tester les solveurs parallèles.

3.2.2 Complexification de la matrice

Considérons un problème générique dépendant du temps et basé sur des équations différentielles ordinaires du premier ordre.

Chaque composante $X_i(t)$ est appelée trajectoire pour insister sur la propriété balistique d'un tel problème complètement déterminé par les conditions initiales. La valeur de n est typiquement dans l'intervalle $[10^3, 10^8]$ de $\mathbb{R}$ et m est le nombre de termes non nuls dans A dans l'intervalle $[10^3,\ 10^9]$; en effet le problème est mieux spécifié par m que par $n * n$.

Initialement la matrice A représente un modèle physique donné dont la solution exacte est inconnue. Le but est d'effectuer des opérations algébriques afin de rendre A moins creuse et afin de tester un solveur parallélisé soumis à une lourde charge de calcul. Pour cela, on considère la première rotation d'un plan construit par deux vecteurs. On appelle la base canonique $E_0 = (e_1, \ldots, e_n)$ et la nouvelle base construite après cette rotation élémentaire $E_\alpha = (e_{\alpha,1}, \ldots, e_{\alpha,n})$, vérifiant

$$\begin{cases} e_{\alpha,1} = \quad \cos(\alpha)e_1 + \sin(\alpha)e_2, \\ e_{\alpha,2} = -\sin(\alpha)e_1 + \cos(\alpha)e_2, \\ e_{\alpha,3} = \quad e_3, \\ \quad \vdots \\ e_{\alpha,n} = \quad e_n. \end{cases}$$

La matrice de changement de base est notée $P_1^{E_0 \to E_\alpha}$. On considère un vecteur $X(t) \in E_0$ avec les composantes $(x_1, .., x_n)$ et le vecteur correspondant $X_\alpha(t) \in E_\alpha$ avec les composantes $(x_{\alpha,1}, .., x_{\alpha,n})$.

On a alors :

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = P_1^{E_0 \to E_\alpha} \begin{pmatrix} x_{\alpha,1} \\ \vdots \\ x_{\alpha,n} \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & & & \\ \sin(\alpha) & \cos(\alpha) & & & 0 \\ & & 1 & & \\ & & & \ddots & \\ & 0 & & & 1 \end{pmatrix} \begin{pmatrix} x_{\alpha,1} \\ \vdots \\ x_{\alpha,n} \end{pmatrix},$$

$$\text{et } \begin{pmatrix} x_{\alpha,1} \\ \vdots \\ x_{\alpha,n} \end{pmatrix} = \left(P_1^{E_0 \to E_\alpha}\right)^{-1} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

Comme la matrice est orthogonale, son inverse est égale à sa transposée :

$$(P_1^{E_0 \to E_\alpha})^t = (P_1^{E_0 \to E_\alpha})^{-1} = P_1^{E_\alpha \to E_0}.$$

Si A est la matrice d'un endomorphisme de E_0 dans E_0 et A_α de E_0 dans E_α alors

$$A_\alpha = (P_1^{E_0 \to E_\alpha})^{-1} A\ P_1^{E_0 \to E_\alpha} = (P_1^{E_0 \to E_\alpha})^t A\ P_1^{E_0 \to E_\alpha}.$$

Maintenant, on considère (3.6) et on multiplie par $(P_1^{E_0 \to E_\alpha})^t$, on a alors successivement

$$\begin{aligned} (P_1^{E_0 \to E_\alpha})^t \dot{X}(t) =& (P_1^{E_0 \to E_\alpha})^t (A\ X(t) + B) \\ =& (P_1^{E_0 \to E_\alpha})^t A\ P_1^{E_0 \to E_\alpha} (P_1^{E_0 \to E_\alpha})^t\ X(t) \\ &+ (P_1^{E_0 \to E_\alpha})^t B. \end{aligned}$$

Le système (3.6) est réécrit dans la base E_α et le système devient

$$\dot{X}_\alpha(t) = A_\alpha X_\alpha(t) + B_\alpha, \tag{3.7}$$

avec $A_\alpha = (P_1^{E_0 \to E_\alpha})^t \; A \; P_1^{E_0 \to E_\alpha}$.

A_α est moins creuse que A, cependant elle n'est pas complètement dense. Pour obtenir une matrice complètement dense, on a besoin de changer de base à nouveau. On considère le changement de base dû à une autre rotation élémentaire d'angle β. On appelle $P_2^{E_\alpha \to E_\beta}$ la matrice de changement de base

$$P_2^{E_\alpha \to E_\beta} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cos(\beta) & -\sin(\beta) & 0 & 0 & \cdots & 0 \\ 0 & \sin(\beta) & \cos(\beta) & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

(*Le plus petit indice correspond à la ligne dans la diagonale de la matrice de rotation 2x2 emboîté dans la grande matrice.*)

Le produit matriciel $P_1^{E_0 \to E_\alpha} P_2^{E_\alpha \to E_\beta}$ est la matrice de changement de base résultant des rotations d'angle α and β.

$P_2^{E_\alpha \to E_\beta}$ est également orthogonale et on a la nouvelle matrice $A_{\alpha\beta}$ telle que

$$A_{\alpha\beta} = (P_1^{E_0 \to E_\alpha} P_2^{E_\alpha \to E_\beta})^t \; A \; P_1^{E_0 \to E_\alpha} P_2^{E_\alpha \to E_\beta}. \tag{3.8}$$

$A_{\alpha\beta}$ est plus dense que A_α. En appliquant d'autres rotations indépendantes on produit une matrice plus dense. Alors il est possible de générer une famille complète de benchmarks pour tester un solveur dans différentes conditions de complexité de modèle. Les couplages et la complexité sont déterminés par le nombre de zéros en dehors de la diagonale.

L'angle de rotation peut être différent pour chaque changement de base. On pourra se référer à la partie 4.4.6 du chapitre 4 pour la preuve que les valeurs propres de A and A_α restent les mêmes en dépit des rotations. Plus d'informations à propos des rotations peuvent être trouvées dans [Bra02].

3.3 Parallélisation avec OpenMP

On a vu les principales caractéristiques de la parallélisation avec OpenMP dans le Chapitre 1 ; on a également vu dans la Section 3.1 quelles étaient les sections de code qui étaient coûteuses et où la parallélisation devait se concentrer et être efficace. Reprenons maintenant en détails ces sections et parallélisons-les avec OpenMP.

3.3.1 Préparation du code

Avant de commencer la parallélisation des différentes sections, il est nécessaire de charger les différentes librairies en utilisant :

```
#include "omp.h"
```

3.3.2 Parallélisation de l'évaluation de la dérivée

L'évaluation de la dérivée de l'ensemble des variables d'état, c'est-à-dire l'évaluation de la fonction f du problème initial coûte 30.9% du coût total. Si on regarde le code de cette fonction f, il s'agit le plus souvent d'une boucle de type for qui calcule successivement les dérivées des différentes variables d'état en fonction du temps et des autres variables d'état. Une parallélisation efficace serait dans ce cas de séparer les itérations de la boucle for selon les différents coeurs alloués. Comme indiqué dans le Chapitre 1, on utilise alors directive

```
#pragma omp parallel for
```

Afin d'envoyer chaque ligne successivement aux différents processeurs dans le but de répartir de façon optimale la charge de travail, on utilise en plus la clause SCHEDULE et imposant des paquets de taille 1 grâce à STATIC :

```
#pragma omp parallel for  schedule(static,1)
```

Dans les cas où la dérivée ne présente pas de boucle for, on utilise alors la directive SECTION plusieurs fois pour évaluer des petits paquets de dérivées des variables d'état. Ces directives SECTION sont utilisées à l'intérieur d'une construction SECTIONS de la manière suivante :

```
#pragma omp parallel
#pragma omp sections
#pragma omp section
...
#pragma omp section
...
#pragma omp end sections
#pragma omp end parallel
```

3.3.3 Parallélisation de la résolution de la partie implicite du schéma LIBDF

On a vu dans la Section 2.2 du Chapitre 2 que le schéma LIBDF permettait d'obtenir un schéma implicite linéaire qu'on peut donc résoudre à l'aide d'un solveur du système linéaire. On utilise un solveur de type gradient biconjugué stabilisé. Le coût de cette résolution est de 23.6% du coût total du programme. Si on effectue un profiling plus complet de ce solveur linéaire qui résout $Ax = B$ on a alors :

- l'ensemble des différents calcul du produit Ax : 57.6%,
- initialisation et copie de différentes variables : 22.7%,
- calcul de normes : 19.7%.

Parallélisation du produit Ax

Le calcul du produit Ax se fait au moyen d'une boucle for sur l'ensemble des lignes de la matrice A (dont le nombre correspond à la taille du problème). Une bonne parallélisation serait de répartir chaque ligne de la matrice A sur différents coeurs au moyen de la directive :

```
#pragma omp parallel for schedule(static,1)
```

Comme précédemment, schedule(static,1) permet de répartir de la façon la plus homogène la charge de travail selon les différents processeurs.

Parallélisation de l'initialisation et copie de différentes variables

L'ensemble de ces calculs se fait au moyen d'une boucle for sur l'ensemble des variables d'état du système ; on peut donc paralléliser ces calculs au moyen de la directive :

```
#pragma omp parallel for
```

Ici on n'utilise pas schedule(static,1) car la charge de travail est la même quelle que soit la variable d'état calculée.

Parallélisation du calcul des normes

Dans ce cas la parallélisation est moins aisée car le calcul de la norme est une opération associative appliquée à une variable (la norme) partagée par l'ensemble des coeurs. Chaque coeur calcule un résultat partiel indépendant des autres et les coeurs se synchronisent à la fin pour déterminer le résultat final. On doit utiliser la clause REDUCTION en précisant la variable partagée et le type d'opération qui lui est appliquée (+,-,*).

```
#pragma omp parallel for section(+,norme)
for(i=0;i<Neq;i++){
norme  = norme + x[i]*x[i];
}
```

3.3.4 Parallélisation de l'initialisation des différentes variables

Cette initialisation coûte 9.5% du temps total de calcul mais est très facilement parallélisable. En effet les variables coûteuses à initialiser sont celles qui sont de même taille que le nombre de variables d'état. Leur initialisation est faite au moyen d'une boucle for sur l'ensemble des variables d'état, on utilise alors, comme précédemment, la directive :

```
#pragma omp parallel for
```

3.3.5 Parallélisation du stockage de la solution

Lorsque la solution a été trouvée, elle est stockée dans un tableau qui enregistre l'ensemble des solutions pour chaque pas de temps. Il est possible de paralléliser ce stockage en enregistrant variable d'état par variable d'état la solution complète. Comme précédemment, on utilise la directive :

```
#pragma omp parallel for
```

3.3.6 Parallélisation du calcul de la partie explicite et de l'interpolation de Lagrange

La partie explicite du schéma LIBDF et l'interpolation de Lagrange sont des combinaisons linéaires des points précédents (et de la dérivée pour le schéma LIBDF). Ces combinaisons linéaires sont identiques pour toutes les variables d'état du système, elles peuvent donc être parallélisées en distribuant les itérations de la boucle for correspondante avec la directive :

```
#pragma omp parallel for
```

3.4 Parallélisation avec MPI

La parallélisation en mémoire distribuée requiert l'utilisation de la bibliothèque de fonctions MPI. Le programme parallèle ne différencie pas beaucoup du programme séquentiel. En effet en dehors des portions du code que l'on cherche à paralléliser toutes les lignes de programme sont exécutées par l'ensemble des processeurs. Les principales fonctions MPI sont décrites dans l'Annexe B. Dans cette section, on va expliquer comment utiliser ces fonctions. Tout d'abord il convient de préparer le code à l'exécution sur plusieurs processeurs en introduisant les lignes de code suivantes :

```
#include "mpi.h"
```

qui permet de charger la librairie contenant les fonctions MPI,

```
MPI_Status status;
```

qui permet d'initialiser la variable status,

```
MPI_Init(&argc, &argv);
```

qui permet l'utilisation des fonctions MPI,

```
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

qui initialise la variable my_rank représentant le numéro de chaque processeur,

```
MPI_Comm_size(MPI_COMM_WORLD, &p);
```

qui initialise la variable p représentant le nombre de processeurs que l'on considère.

A l'instant t_i, tous les processeurs ont les solutions X_{i-p} entières stockées dans leur propre mémoire ainsi que la matrice Jacobienne entière également.

Au début de chaque itération en temps on commence par calculer l'interpolation et la partie explicite du schéma en utilisant en particulier des appels à la dérivée et à la matrice Jacobienne.

3.4.1 Parallélisation du calcul de l'interpolation

Le calcul de l'interpolation n'est (dans le cas de l'interpolation polynomiale) qu'une combinaison linéaire des points précédents. Ainsi le premier processeur ne calcule que les n/p premières variables de cette estimation (n est la taille du système), le second les variables comprises entre la $n/p+1$-ième et la $2n/p$ et ainsi de suite. Ainsi la taille de la boucle for nécessaire pour chaque processeur au calcul de l'interpolation décrite dans la Section 2.2.7 est divisée par p.

3.4.2 Parallélisation du calcul de la dérivée

Le calcul de la dérivée qui correspond à l'évaluation de la fonction f est répartie sur les différents processeurs de la même manière que le calcul de l'interpolation. Ainsi le processeur 1 calcule les n/p premières dérivées, le processeur 2 les n/p suivantes et ainsi de suite.

3.4.3 Calcul du produit matrice vecteur de la matrice Jacobienne et de l'interpolation

Afin de répartir le produit de la matrice Jacobienne et de l'interpolation sur chaque processeur, il est nécessaire que tous les processeurs aient en mémoire l'intégralité du vecteur de l'interpolation. En effet, chaque processeur n'a calculé que n/p composantes de ce vecteur et on utilise alors la fonction MPI_ALLGATHER (décrite dans la Section 1.2.3 du Chapitre 1) afin que chaque processeur envoie à tous les autres sa partie de vecteur et reçoit l'ensemble du vecteur. Une fois que chaque processeur a bien reçu l'intégralité du vecteur d'interpolation, chaque processeur effectue le produit matrice Jacobienne vecteur d'interpolation en utilisant seulement une bande de la matrice Jacobienne contenant n/p lignes et ainsi chaque processeur obtient une partie du vecteur résultat de taille n/p.

3.4.4 Calcul de la partie explicite du schéma LIBDF

La dérivée étant calculée, le produit matrice Jacobienne vecteur d'interpolation effectué, chaque processeur peut maintenant calculer la partie explicite du schéma LIBDF qui est une combinaison linéaire des points précédents (stockés dans chaque processeur), de la dérivée et du produit Jacobienne-Interpolation. Chaque processeur ne calcule que n/p composantes de cette partie explicite. Une fois que ce calcul est fait, on utilise à nouveau la fonction MPI_ALLGATHER afin que l'ensemble des processeurs envoient le morceau de partie explicite à tous les autres processeurs et qu'ils reçoivent l'ensemble de cette partie explicite.

3.4.5 Calcul de l'inversion du système linéaire

L'inversion du système linéaire complet se fait simultanément par chaque processeur, la matrice d'inversion étant stockée sur chaque processeur ainsi que le second membre qui correspond à la partie explicite du schéma LIBDF. Ainsi la solution est déterminée pour le temps t_{i+1} et chaque processeur a donc en mémoire la solution complète en ce temps et la résolution peut donc se poursuivre pour les temps suivants.

3.5 Parallélisation hybride

Il est possible d'effectuer une parallélisation hybride en mêlant fonctions MPI et directives OpenMP. En effet, si on utilise une architecture multiprocesseurs où chaque processeur a plusieurs coeurs de calcul, on peut paralléliser le calcul à l'intérieur de chaque processeur. Dans un premier temps, on parallélise selon les différents processeurs en utilisant les fonctions MPI décrites dans la Section 3.4. Puis dans chaque processeur on parallélise en utilisant les directives OpenMP décrites dans la Section 3.3. Ainsi chaque boucle qui était de taille n/p sera à nouveau divisée en boucles plus petites et la résolution du système linéaire qui n'était pas parallélisée se trouve ainsi parallélisée.

Chapitre 4

Applications issues de Modelica

Ce chapitre présente quatre cas d'application des méthodes d'accélération décrites dans les chapitres précédents. Le premier cas est un modèle hybride qui permet de tester la méthode de détection des évènements décrite dans la Section 2.4 du Chapitre 2. Le deuxième et troisième cas sont des modèles de grande taille non linéaires permettant de comparer l'efficacité du schéma LIBDF par rapport au schéma BDF en terme de rapidité d'exécution mais également en terme d'erreur. Le dernier cas est un modèle de diffusion de la chaleur exécuté en parallèle et utilisant la méthode de densification décrit dans la Section 3.2 du Chapitre 3.

Sommaire

4.1 Cas de la balle rebondissante

Afin de montrer la méthode de gestion du temps hybride décrite dans la Section 2.4 du Chapitre 2, on utilise l'exemple de la balle rebondissante comme benchmark. Considérons tout d'abord la balle rebondissante dans le contexte des sphères dures. En d'autres termes, la forme de la balle ne change pas lorsqu'elle frappe le sol. En ce qui concerne les conditions initiales, la balle est jetée en l'air avec une vitesse initiale non nulle.

4.1.1 Equations avec frottements de l'air

Considérons d'abord la trajectoire balistique de la balle avant qu'elle ne touche le sol. On applique le principe fondamental de la dynamique à cette balle, considérée comme une masse ponctuelle m d'accélération $\boldsymbol{a}$ soumise à la somme des forces $\sum_i \boldsymbol{F_i} = m\boldsymbol{a}$

De plus, on considère que les frottements de l'air sont proportionnels au carré de la norme du vecteur vitesse et de sens opposé [TvdW99]. Avec C_x le coefficient de trainée, S la surface de section de la balle, ρ_{air} la densité de l'air, l'expression de la force de frottements est

$$\boldsymbol{F}_{drag} = -\frac{1}{2} C_x \rho_{air} S ||\boldsymbol{v}|| \boldsymbol{v}.$$

En écrivant le vecteur vitesse dans un repère cartésien 2D (0,$\boldsymbol{u_x}$,$\boldsymbol{u_y}$) comme $\boldsymbol{v} = v_x\ \boldsymbol{u_x} + v_y\ \boldsymbol{u_y}$, on obtient :

$$\boldsymbol{F}_{drag} = -\frac{1}{2} C_x \rho_{air} S ||\boldsymbol{v}|| (v_x\ \boldsymbol{u_x} + v_y\ \boldsymbol{u_y})$$

avec $||\boldsymbol{v}|| = \sqrt{v_x^2 + v_y^2}$. Durant sa trajectoire balistique, la balle est soumise à son poids et aux frottements de l'air. Le principe fondamental de la dynamique peut alors s'écrire sous la forme

$$m\ \frac{\mathrm{d}}{\mathrm{d}t}(v_x\ \boldsymbol{u_x} + v_y\ \boldsymbol{u_y}) = -m\ g\ \boldsymbol{u_y} - \frac{1}{2} C_x \rho_{air} S ||\boldsymbol{v}|| (v_x\ \boldsymbol{u_x} + v_y\ \boldsymbol{u_y}). \tag{4.1}$$

En projetant la dernière équation dans le repère cartésien (0,$\boldsymbol{u_x}$,$\boldsymbol{u_y}$) on a un système d'équations différentielles ordinaires :

$$\begin{cases} m\dfrac{\mathrm{d}}{\mathrm{d}t} v_x = -\dfrac{1}{2} C_x \rho_{air} S ||\boldsymbol{v}|| v_x, \\ m\dfrac{\mathrm{d}}{\mathrm{d}t} v_y = -m\ g\ - \dfrac{1}{2} C_x \rho S ||\boldsymbol{v}|| v_y, \\ \dfrac{\mathrm{d}x}{\mathrm{d}t} = v_x, \qquad \dfrac{\mathrm{d}y}{\mathrm{d}t} = v_y, \\ v_x(t=0) = v_{x0}, \qquad v_y(t=0) = v_{y0}, \\ x(t=0) = x_0, \qquad y(t=0) = y_0, \end{cases} \tag{4.2}$$

On a 4 variables décrivant explicitement en un temps donné la trajectoire balistique de la balle, qui sont :

- les composantes verticales et horizontales de la vitesse v_x et v_y respectivement,
- les deux coordonnées x et y de la balle dans le plan (0,$\boldsymbol{u_x}$,$\boldsymbol{u_y}$).

4.1.2 Les équations de la dynamique sans frottements de l'air

Considérons l'exemple précédent sans frottements de l'air.

$$\begin{cases} m\dfrac{\mathrm{d}}{\mathrm{d}t}v_x = 0, \\ m\dfrac{\mathrm{d}}{\mathrm{d}t}v_y = -m\ g\ , \\ \dfrac{\mathrm{d}x}{\mathrm{d}t} = v_x, \qquad \dfrac{dy}{dt} = v_y, \\ v_x(t=0) = v_{x0}, \qquad v_y(t=0) = v_{y0}, \\ x(t=0) = x_0, \qquad y(t=0) = y_0, \end{cases} \tag{4.3}$$

On résout facilement (4.3) en intégrant les deux premières équations deux fois :

$$\begin{aligned} x(t) &= v_{x0}t + x_0, \\ y(t) &= -m\ g\ t^2 + v_{y0}t + y_0. \end{aligned}$$

On remarque que les solutions $x(t)$ et $y(t)$ sont respectivement une fonction linéaire et une parabole. Cependant le modèle avec frottements est plus raide, non linéaire et physiquement plus réaliste, c'est celui-ci qui sera choisi dans la suite de la Section.

4.1.3 Résultats numériques

On utilise les valeurs numériques suivantes :

$$\begin{aligned} v_{x0} &= 1\ m/s \qquad v_{y0} = 5\ m/s \\ x_0 &= 0\ m \qquad\quad y_0 = 2\ m \\ C_x &= 0.5 \\ R &= 0.1\ m \\ \rho_{air} &= 1.293\ kg/m^3 \qquad \rho_{balle} = 500\ kg/m^3 \\ g &= 9.81\ m/s^2 \end{aligned}$$

Et pour la Section 4.1.13 :

$$\epsilon = 0.1, \quad E = 0.1\ GPa. \tag{4.4}$$

Sur la Figure 4.1, on trace un rebond de la balle avec frottements de l'air (ligne en pointillé) et sans frottements de l'air (ligne continue).

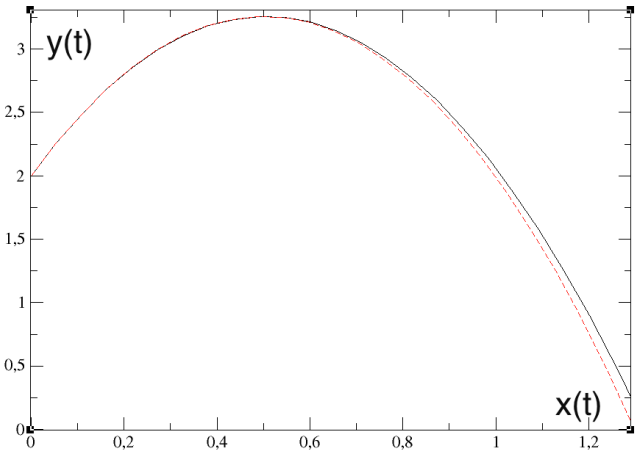


Figure 4.1 – Trajectoire de balle avec et sans frottements de l'air.

4.1.4 Utilisation d'un vecteur de contraintes

Dans l'exemple de la balle rebondissante, l'évènement "la balle touche le sol" correspond à la contrainte "le signe de la position verticale de la balle change". Le vecteur des contraintes est fait d'un seul scalaire, la hauteur de la balle par rapport au sol $y(t) - y_{ground}$. Dès que le signe de la composante change, l'évènement se produit. Pour l'exemple, on écrit, en appelant y_{ground} la hauteur par rapport au sol :

$$\boldsymbol{c} = [c_0] = [y - y_{ground}] < 0.$$

4.1.5 Expression du système avec les contraintes

En considérant le système (4.2) et en ajoutant les contraintes (on choisit ici $y_{ground} = 0$), on obtient le système complet avec contraintes vérifiées par la balle rebondissante.

$$\begin{cases} m\dfrac{\mathrm{d}}{\mathrm{d}t}v_x = -\dfrac{1}{2}C_x\rho_{air}S||\boldsymbol{v}||v_x, \\ m\dfrac{\mathrm{d}}{\mathrm{d}t}v_y = -m\ g\ -\dfrac{1}{2}C_x\rho_{air}S||\boldsymbol{v}||v_y, \\ \dfrac{\mathrm{d}x}{\mathrm{d}t} = v_x, \qquad \dfrac{dy}{dt} = v_y, \\ v_x(t=0) = v_{x0}, \qquad v_y(t=0) = v_{y0}, \\ x(t=0) = x_0, \qquad y(t=0) = y_0, \\ c = y(t) > 0. \end{cases}$$

4.1.6 Calcul du temps de l'évènement

On va illustrer la méthode décrite dans la Section 2.4.3 avec l'exemple de la balle rebondissante. Dans ce cas, on ne peut pas détecter l'instant précis au moment où la balle touche le sol avec seulement les noeuds comme le rebond se produit entre les noeuds. Nous saurons que la balle a traversé le sol uniquement quand la solution suivante aura une composante verticale négative.

Sur la Figure 4.2, on trace l'ordonnée de la position de la balle comme une fonction du temps. Le dernier intervalle d'intégration traverse le sol. On met en avant l'instant où la trajectoire balistique de la balle traverse le sol comme un évènement et la détermination de cet évènement nécessite de couper ce dernier intervalle.

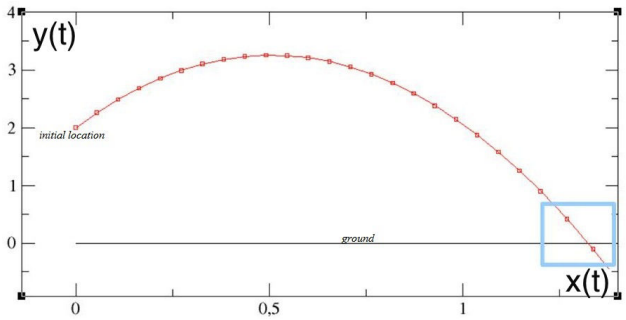


FIGURE 4.2 – Ordonnée de la balle en fonction du temps. Durant le dernier intervalle la balle passe à travers le sol.

Afin de connaître les coordonnées du point d'intersection de la courbe précédente avec $y = 0$, il est possible d'utiliser une dichotomie sur l'intervalle jusqu'à ce que l'on trouve le temps t^* pour lequel l'ordonnée de la balle est nulle [ZYM08]. Cette méthode coûte cher car elle nécessite de nombreux calculs de $\dot{y} = f(y)$. On suggère dans la Section 2.4.3 une autre méthode qui coûte moins cher pour calculer t^*, que l'on illustre avec la balle rebondissante.

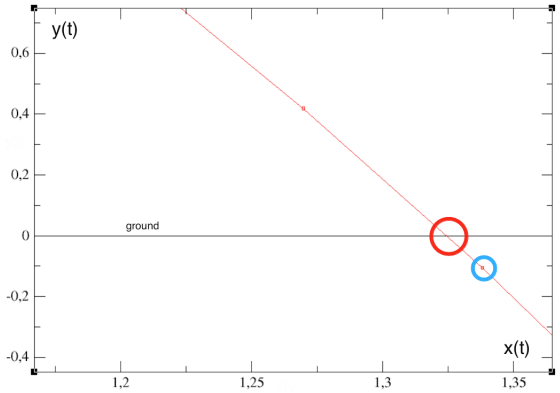


FIGURE 4.3 – Altitude de la balle en fonction du temps quand la trajectoire balistique traverse le sol.

Dans l'exemple de balle rebondissante, on interpole un polynôme du second degré en temps sur les trois derniers points de la trajectoire (voir figure 4.4). Ensuite on calcule analytiquement les racines et on choisit celle située dans le bon intervalle. On obtient alors le temps t^* quand la contrainte est égale à zéro.

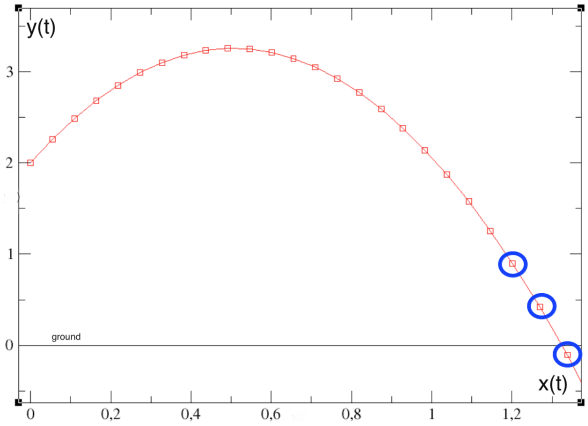


Figure 4.4 – Interpolation par un polynôme du second degré en temps basé sur les trois points en bleu.

4.1.7 Calcul de la solution en t^*

On calcule la solution en t^* afin de redémarrer la résolution du système en utilisant comme nouvelle condition initiale cette solution en t^*. On utilise la même méthode que la méthode décrite dans la Section 2.4.4 et on représente l'interpolation sur la Figure 4.5.

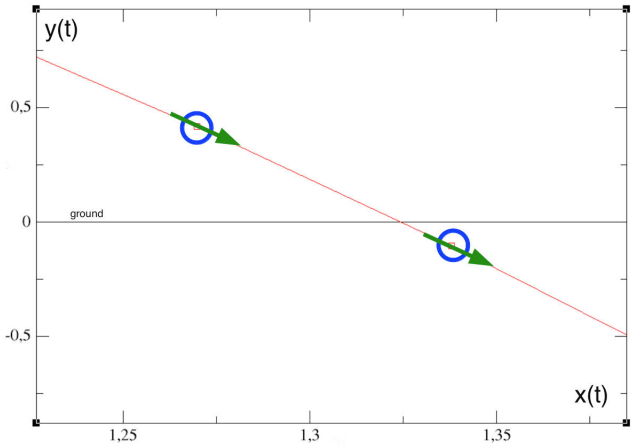


Figure 4.5 – Interpolation au moyen d'un polynôme du troisième degré basé sur les deux points et leurs tangentes.

4.1.8 Résolution du système après un évènement

Dans l'exemple particulier de la balle rebondissante dans le cas des sphères dures, les équations ne changent pas, seules les conditions initiales de la composante verticale de la vitesse sont modifiées et prennent une valeur opposée à t^*, multipliée par un facteur d'amortissement [LST01] modélisant l'énergie perdue durant le rebond.

Le système d'équations rassemblées avec $x^*, y^*, v_{x^*}, v_{y^*}$ comme conditions initiales, produit un nouveau système dynamique qui doit être intégré jusqu'au rebond suivant [Mos99].

$$\begin{cases} m\dfrac{\mathrm{d}}{\mathrm{d}t}v_x = -\dfrac{1}{2}C_x\rho_{air}S||\boldsymbol{v}||v_x, \\ m\dfrac{\mathrm{d}}{\mathrm{d}t}v_y = -m\ g\ -\dfrac{1}{2}C_x\rho_{air}S||\boldsymbol{v}||v_y, \\ \dfrac{\mathrm{d}x}{\mathrm{d}t} = v_x, \qquad \dfrac{\mathrm{d}y}{\mathrm{d}t} = v_y, \\ v_x(t=t^*) = v_{x^*}, \qquad v_y(t=t^*) = -v_{y^*}(1-\epsilon), \\ x(t=t^*) = x^*, \qquad y(t=t^*) = y^*, \\ c = y(t). \end{cases}$$

Comme le modèle ou les conditions initiales sont modifiés, il est important de redémarrer l'intégrateur numérique à l'évènement en ne considérant plus les points calculés précédemment.

4.1.9 Balle dure rebondissante sur un sol plat

On représente la trajectoire balistique de la balle dans le plan 2D en traçant l'ordonnée $y(t)$ comme une fonction de l'abscisse $x(t)$.

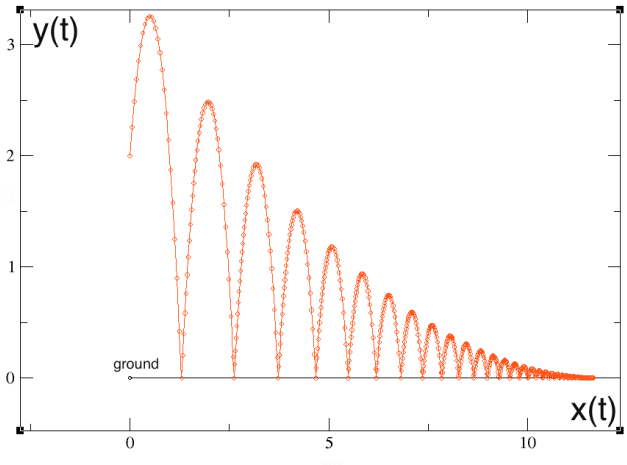


FIGURE 4.6 – Trajectoire de la balle rebondissante sur un sol plat.

On va maintenant comparer la méthode de gestion des évènements précédemment décrite avec la méthode utilisée par le solveur DASSL d'OpenModelica. Pour cela, on introduit la méthode de gestion des évènements dans un solveur utilisant un schéma de type BDF dont les propriétés sont décrites dans le Tableau 4.2.

Caractéristiques du modèle

Non linéaire	Type de nonlinéarité	Taille	Sparsité	Evènements
Oui	Ordre 2 par rapport à la vitesse	2	0	Oui (rebonds)

TABLE 4.1 – Description du modèle.

Caractéristiques du solveur utilisé

Schéma	Ordre	Pas de temps	Tolérance	Parallélisme	Nombre de tâches
BDF	6	Variable	10^{-6}	Non	1

Table 4.2 – Description du solveur utilisé.

Code Modelica

Le modèle de la balle rebondissante est le suivant :

```
model BouncingBall
  Real vx(start = 1);
  Real vy(start = 5);
  Real x(start = 0);
  Real y(start = 2);
  parameter Real m = 1.1;
  parameter Real Cx = 0.5;
  parameter Real rho = 1.293;
  parameter Real S = 3.14 * 0.1 * 0.1;
  constant Real g = 9.81;
equation
  m * der(vx) = -0.5 *Cx * rho * S *
                        sqrt(vx ^ 2 + vy ^ 2) * vx;
  m * der(vy) = -m * g - 0.5 * Cx * rho * S *
                        sqrt(vx ^ 2 + vy ^ 2) * vy;
  der(x) = vx;
  der(y) = vy;
  when y <= 0 then
      reinit(vy, -0.9 * pre(vy));
  end when;
end BouncingBall;
```

Comparaison avec DASSL

Les résultats donnés par le solveur décrit dans la Table 4.2 et qui utilise la méthode de gestion des évènements décrit précédemment sont comparés avec les résultats donnés par un autre solveur implémenté dans OpenModelica, DASSL.

Comparons le temps CPU pour la simulation de la balle rebondissante durant 20 secondes avec DASSL [Pet82] et le solveur utilisé. La tolérance relative est 10^{-6} et le coefficient de friction de l'air est $C_x \rho S = 0.0203\ kg/m$. Sur le tableau 4.3, les temps CPU sont divisés par deux.

Temps CPU DASSL	Temps CPU solveur utilisé
0.068s	0.032s

Table 4.3 – Temps d'exécution.

La variable y tracée pour les deux solveurs est représentée sur la Figure 4.7.

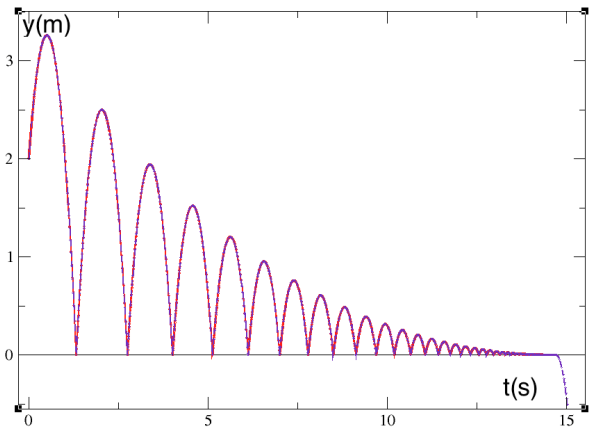


FIGURE 4.7 – Résultats de DASSL en bleu et en pointillé, le solveur utilisé en rouge et en ligne continue

Un zoom du point de rebond (le dixième) aux alentours de $9.12s$ montre la différence entre les deux solutions qui restent très proches, sur la Figure 4.8.

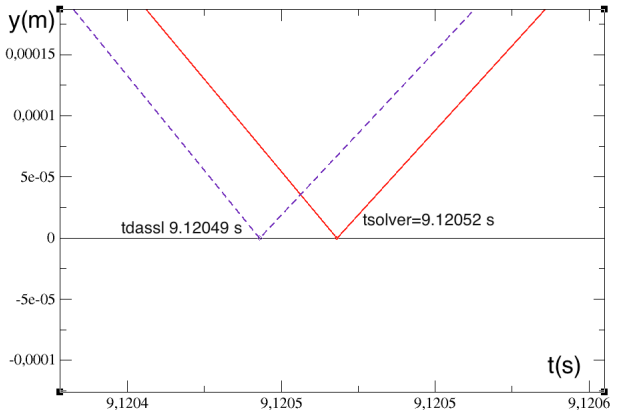


FIGURE 4.8 – DASSL en bleu et en pointillé, le solveur utilisé en rouge et en continu.

Maintenant on trace la différence absolue des temps d'évènements en DASSL et le solveur utilisé pour les dix premiers rebonds sur la Figure 4.9.

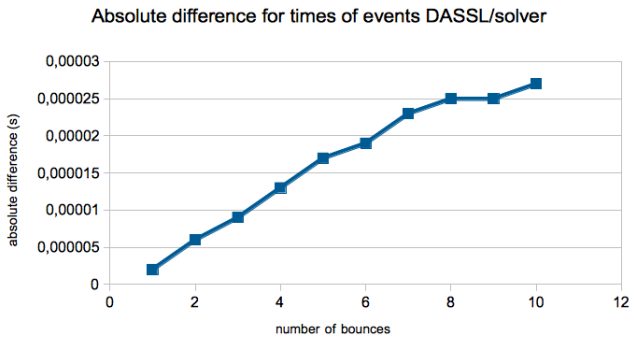


Figure 4.9 – Différence des temps d'évènements entre DASSL et le solveur utilisé.

L'augmentation de la différence est pratiquement linéaire mais la différence relative, elle n'augmente pratiquement plus après plusieurs rebonds comme on peut le voir sur la Figure 4.10.

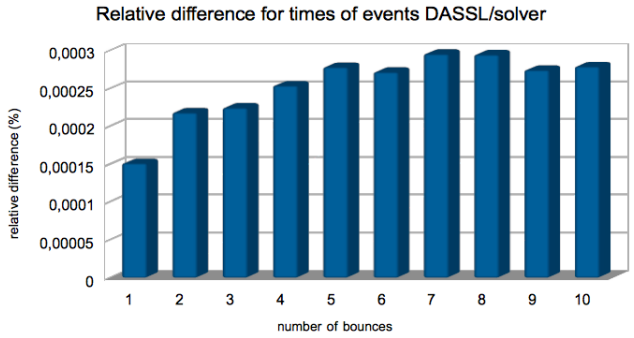


Figure 4.10 – Différence des temps d'évènements entre DASSL et le solveur utilisé.

Comparaison avec la solution analytique :

Afin de comparer la solution donnée par DASSL ou par le solveur avec la solution analytique, on considère le problème de la balle rebondissante à une seule dimension.

Calcul de la solution analytique : le système suivant est utilisé comme point de départ.

$$\begin{cases} m\dfrac{\mathrm{d}}{\mathrm{d}t}v_y = -m\ g\ - \dfrac{1}{2}C_x\rho_{air}S|v_y|v_y, \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = v_y, \\ v_y(t=0) = 0, \\ y(t=0) = y_0. \end{cases} \tag{4.5}$$

Tout d'abord on considère la trajectoire de la balle avant le premier rebond, alors $|v_y| = -v_y$. La première équation du système s'écrit :

$$\begin{aligned} m\frac{\mathrm{d}}{\mathrm{d}t}v_y &= -mg\ + \frac{1}{2}C_x\rho_{air}Sv_y^2 \\ \frac{\mathrm{d}}{\mathrm{d}t}v_y - \frac{C_x\rho_{air}S}{2m}v_y^2 &= -g. \end{aligned}$$

C'est une équation de Riccati résolue au moyen d'un changement de variable $\beta = \frac{C_x \rho_{air} S}{2m}$

$$v_y(t) = \frac{\lambda(t)}{1 - \beta \int_0^t \lambda(s)ds}. \tag{4.6}$$

En dérivant (4.6) il vient :

$$v_y'(t) = \frac{\lambda'(t)}{1 - \beta \int_0^t \lambda(s)ds} + \beta \left(\frac{\lambda(t)}{1 - \beta \int_0^t \lambda(s)ds} \right)^2$$
$$v_y'(t) - \beta v_y(t)^2 = \frac{\lambda'(t)}{1 - \beta \int_0^t \lambda(s)ds} = -g.$$

L'équation à résoudre est maintenant :

$$\lambda'(t) = -g \left(1 - \beta \int_0^t \lambda(s)ds \right). \tag{4.7}$$

EN posant $G'(t) = \lambda(t)$, on obtient :

$$G''(t) = -g(1 - \beta G) \tag{4.8}$$

$$\left(G - \frac{1}{\beta} \right)'' - \beta g \left(G - \frac{1}{\beta} \right) = 0 \tag{4.9}$$

(4.9) est une équation différentielle du second ordre et comme $4\beta g > 0$, la solution de (4.9) est

$$G(t) = A \exp(\sqrt{\beta g} t) + B \exp(-\sqrt{\beta g} t) + \frac{1}{\beta}, \tag{4.10}$$

et

$$v_y(t) = \frac{G'(t)}{1 - \beta G(t)}$$
$$= \frac{\sqrt{\beta g}(-A \exp(\sqrt{\beta g} t) + B \exp(-\sqrt{\beta g} t))}{\beta(A \exp(\sqrt{\beta g} t) + B \exp(-\sqrt{\beta g} t))}$$

En intégrant $v_y(t)$ on obtient l'expression de $y(t)$:

$$y(t) = -\frac{1}{\beta} \ln \left(A \exp(\sqrt{\beta g} t) + B \exp(-\sqrt{\beta g} t) \right) \tag{4.11}$$

A et B sont déterminés grâce aux conditions initiales

Quand la balle touche le sol, le signe de v_y change ($|v_y(t)| = v_y(t)$) et la première équation de (4.5) se réécrit :

$$m \frac{\mathrm{d}}{\mathrm{d}t} v_y = -m\ g\ - \frac{1}{2} C_x \rho_{air} S v_y^2$$
$$\frac{\mathrm{d}}{\mathrm{d}t} v_y + \frac{C_x \rho_{air} S}{2m} v_y^2 = -g.$$

Il s'agit toujours d'une équation de Riccati résolue au moyen d'un changement de variable (avec $\beta = \frac{C_x \rho_{air} S}{2m}$) :

$$v_y(t) = \frac{\mu(t)}{1 + \beta \int_0^t \mu(s)ds}. \tag{4.12}$$

En dérivant (4.12) il vient :

$$v'_y(t) = \frac{\mu'(t)}{1+\beta\int_0^t \mu(s)ds} - \beta\left(\frac{\mu(t)}{1+\beta\int_0^t \mu(s)ds}\right)^2$$
$$v'_y(t) + \beta v_y(t)^2 = \frac{\mu'(t)}{1+\beta\int_0^t \mu(s)ds} = -g.$$

L'équation à résoudre est :

$$\mu'(t) = -g\left(1+\beta\int_0^t \mu(s)ds\right). \tag{4.13}$$

Avec $H'(t) = \mu(t)$, on trouve

$$H''(t) = -g(1+\beta H) \tag{4.14}$$
$$\left(H(t)+\frac{1}{\beta}\right)'' + \beta g\left(H(t)+\frac{1}{\beta}\right) = 0 \tag{4.15}$$

(4.15) est une équation différentielle du second ordre et comme $4\beta g > 0$ la solution de (4.15) est :

$$H(t) = C\cos(\sqrt{\beta g}t) + D\sin(\sqrt{\beta g}t) - \frac{1}{\beta}. \tag{4.16}$$

et

$$v_y(t) = \frac{H'(t)}{1+\beta H(t)}$$
$$v_y(t) = \frac{\sqrt{\beta g}(-C\sin(\sqrt{\beta g}t) + D\sin(\sqrt{\beta g}t))}{\beta(C\cos(\sqrt{\beta g}t) + D\sin(\sqrt{\beta g}t))}$$

En intégrant $v_y(t)$ on obtient l'expression de $y(t)$

$$y(t) = \frac{1}{\beta}\ln\left(C\cos(\sqrt{\beta g}t) + D\sin(\sqrt{\beta g}t)\right) \tag{4.17}$$

C et D sont déterminés avec les valeurs de la précédente solution au moment où la balle touche le sol. La suite de la solution est obtenue en traçant la bonne fonction selon que la balle monte ou non. Les temps du premier et du second évènement sont donnés par la solution analytique, ainsi DASSL et le solveur utilisé peuvent être comparés.
La Table 4.4 montre pour deux tolérances relatives que le solveur utilisé détecte plus précisément les évènements que DASSL et de plus avec un temps de simulation plus petit. La tolérance relative indiquée est valide uniquement pour les points calculés par le solveur pendant la trajectoire en l'air mais non pour les points d'évènements qui doivent être calculés très précisément. En conséquence, ce résultat valide la gestion d'évènements décrite plus haut.

Nombre de rebonds	Tolérance relative	Solution analytique	DASSL	Erreur DASSL	Solveur utilisé	Erreur solveur utilisé
1er	10^{-6}	0.640714	0.640715	$1,56.10^{-6}$	0.640714	0
	10^{-3}	0.640714	0.641162	7.10^{-4}	0.640730	$2,50.10^{-5}$
2nd	10^{-6}	1.769519	1.769523	$2,26.10^{-6}$	1.769515	$2,26.10^{-6}$
	10^{-3}	1.769519	1.769915	$2,23.10^{-4}$	1.769562	$2,43.10^{-5}$

Table 4.4 – Comparaison des temps du premier évènement.

La différence est très petite pour le premier rebond ; cependant, les écarts sont cumulés d'un rebond à l'autre.

4.1.10 Balle dure rebondissante sur un sol sinusoïdal

On va maintenant comparer la méthode de gestion des évènements décrite précédemment sur un sol sinusoïdal avec une solution qui se rapproche le plus possible de la solution analytique. De même que précédemment, la méthode de gestion des évènements est introduite dans un solveur utilisant un schéma de type BDF dont les propriétés sont données dans le Tableau 4.7.

Caractéristiques du modèle

Non linéaire	Type de nonlinéarité	Taille	Sparsité	Evènements
Oui	Ordre 2 par rapport à la vitesse	2	0	Oui (rebonds)

Caractéristiques du solveur utilisé

Schéma	Ordre	Pas de temps	Tolérance	Parallélisme	Nombre de tâches
BDF	6	Variable	10^{-6}	Non	1

On suppose que le sol n'est plus plat mais devient sinusoïdal. On appelle $gr(x(t))$ la hauteur du sol par rapport à l'origine 0, que l'on définit par

$$gr(x(t)) = gr_0 \ \cos(\omega x(t)).$$

Ensuite, la contrainte c dont le signe doit être verifié, peut être écrite :

$$c(t) = y(t) - gr_0 \ \cos(\omega x(t)).$$

Calcul de la balle rebondissante sur un sol sinusoïdal :

$$\frac{\mathrm{d}gr}{\mathrm{d}x} = gr'(x(t)) = -\omega gr_0 \ \sin(\omega x(t)).$$

Dans le repère de Frénet associé à la balle au moment où elle heurte le sol, on obtient les expressions des vecteurs $\boldsymbol{T}$ et $\boldsymbol{N}$ de la trajectoire

$$\boldsymbol{T} = \frac{1}{||\boldsymbol{T}||}\begin{pmatrix} 1 \\ gr'(t) \end{pmatrix}, \quad \boldsymbol{N} = \frac{1}{||\boldsymbol{N}||}\begin{pmatrix} -gr'(t) \\ 1 \end{pmatrix}.$$

Dans le repère cartésien du plan $(0, \boldsymbol{u_x}, \boldsymbol{u_y})$, on a

$$\boldsymbol{N} = \frac{\omega \ gr_0 \ \sin(\omega x)\boldsymbol{u_x} + \boldsymbol{u_y}}{\sqrt{1+\omega^2 gr_0^2 \sin^2(\omega x)}}, \quad \boldsymbol{T} = \frac{\boldsymbol{u_x} - \omega \ gr_0 \ \sin(\omega x)\boldsymbol{u_y}}{\sqrt{1+\omega^2 gr_0^2 \sin^2(\omega x)}}.$$

En considérant $\boldsymbol{N} - \omega \ gr_0 \ \sin(\omega x)\boldsymbol{T}$, on en déduit l'expression de $\boldsymbol{u_x}$ et $\boldsymbol{u_y}$

$$\boldsymbol{u_x} = \frac{\omega \ gr_0 \ \sin(\omega x)\boldsymbol{N} + \boldsymbol{T}}{\sqrt{1+\omega^2 gr_0^2 \sin^2(\omega x)}}, \quad \boldsymbol{u_y} = \frac{\boldsymbol{N} - \omega \ gr_0 \ \sin(\omega x)\boldsymbol{T}}{\sqrt{1+\omega^2 gr_0^2 \sin^2(\omega x)}}.$$

Le vecteur vitesse est exprimé dans le repère cartésien $(0, \boldsymbol{u_x}, \boldsymbol{u_y})$ comme $\boldsymbol{v} = v_x\boldsymbol{u_x} + v_y\boldsymbol{u_y}$

Connaissant l'expression de $\boldsymbol{u_x}$ et de $\boldsymbol{u_y}$ comme fonctions de $\boldsymbol{N}$ et $\boldsymbol{T}$, on exprime $\boldsymbol{v}$ dans le repère de Frénet

$$\boldsymbol{v} = \frac{(v_x\omega gr_0\sin(\omega x) + v_y)\boldsymbol{N}}{\sqrt{1+\omega^2 gr_0^2\sin^2(\omega x)}} + \frac{(v_x - \omega gr_0\sin(\omega x)v_y)\boldsymbol{T}}{\sqrt{1+\omega^2 gr_0^2\sin^2(\omega x)}}.$$

D'après les lois de Descartes, on sait que la balle rebondit avec un angle opposé par rapport à la normale du sol et avec un facteur d'amortissement. L'expression de la balle après le rebond est :

$$\boldsymbol{v}^* = \frac{-(1-\epsilon)(v_x\omega gr_0\sin(\omega x) + v_y)\boldsymbol{N}}{\sqrt{1+\omega^2 gr_0^2\sin^2(\omega x)}} + \frac{(v_x - \omega gr_0\sin(\omega x)v_y)\boldsymbol{T}}{\sqrt{1+\omega^2 gr_0^2\sin^2(\omega x)}}.$$

Pour revenir à l'expression dans le repère cartésien, on remplace $\boldsymbol{N}$ et $\boldsymbol{T}$ par leurs expressions comme fonctions de $\boldsymbol{u_x}$ et $\boldsymbol{u_y}$. Finalement, on obtient l'expression de la nouvelle vitesse

$$\boldsymbol{v}^* = \frac{(-\omega^2 sol_0^2\sin^2(\omega x)(1-\epsilon)+1)v_x\boldsymbol{u_x}}{1+\omega^2 gr_0^2\sin^2(\omega x)} + \frac{(-\beta+\omega^2 gr_0^2\sin^2(\omega x))v_y\boldsymbol{u_y}}{1+\omega^2 gr_0^2\sin^2(\omega x)} \tag{4.18}$$

Si on trace la trajectoire de la balle, on obtient la courbe suivante sur la Figure 4.11 ($\omega = \pi$, $gr_0 = 0.05$, $\epsilon = 0.2$, $v_{x0} = 5$,$v_{y0} = 5$,$x_0 = 0$ and $y_0 = 1.5$).

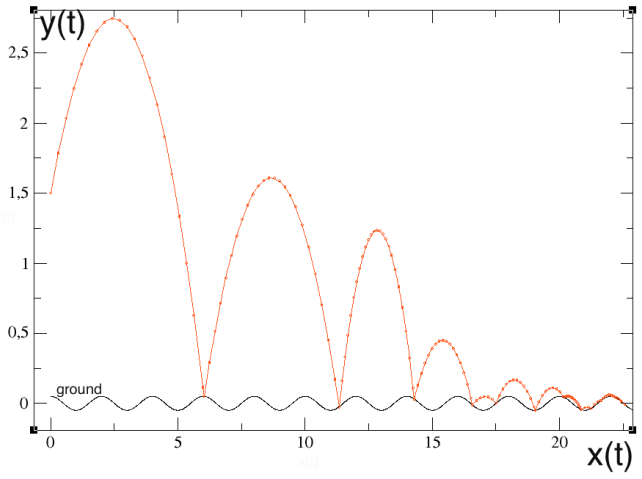


Figure 4.11 – Trajectoire de la balle rebondissante sur un sol sinusoïdal.

En augmentant gr_0 et la période, la trajectoire reste bloquée dans un creux comme on peut le voir sur la Figure 4.12 (avec $\omega = \frac{2\pi}{10}$, $gr_0 = 1$ et $\epsilon = 0.2$)

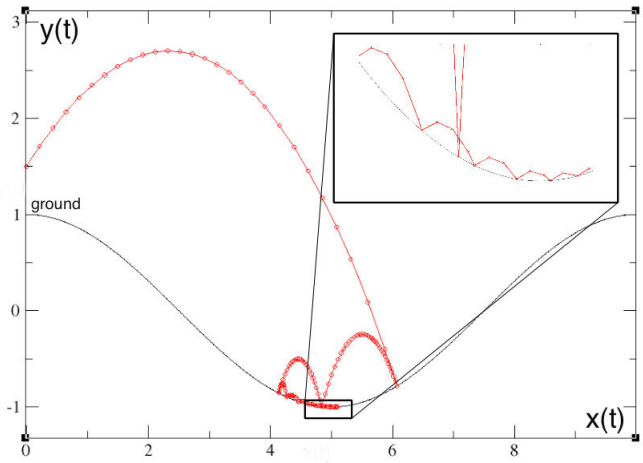


FIGURE 4.12 – Balle rebondissante sur un sol sinusoïdal.

4.1.11 Benchmarking

On teste la méthode précédente de calcul de X^* en comparant la solution que l'on obtient avec la valeur théorique. Comme nous ne connaissons pas la solution analytique on utilise un schéma d'Euler sans traitement d'évènement, avec un pas fixe de 10^{-7} secondes. Avec ce schéma et un calcul coûteux (13008621 points), on connaît avec précision la trajectoire de la balle quand elle traverse le sol. Ensuite, on compare la position théorique de la balle en utilisant ce schéma d'Euler et la position de la balle obtenue au moyen de la gestion des évènements en utilisant des polynômes.

Un paramètre du schéma numérique que nous utilisons est la tolérance qui est l'erreur maximale permise pour chaque point. Modifions maintenant la tolérance du modèle et comparons le nombre de points à partir de la position initiale jusqu'au premier rebond et la distance relative entre le point de rebond théorique et le point de rebond effectif. On remarque que pour une tolérance plus petite les calculs nécessitent plus de points mais sont plus précis.

Tolérance	Nombre de points	Distance relative
1%	3	0,048%
0.1%	8	0,0061%
0.01%	18	0,000077%

TABLE 4.5 – Résultats du schéma numérique pour différentes tolérances.

4.1.12 Limites du modèle

La technique principale de détection du zero-crossing de la contrainte est liée à des hypothèses. On a besoin d'au moins 3 points entre chaque rebond pour être capable d'interpoler le polynôme et ainsi de détecter le changement de signe de la contrainte. En conséquence, quand les évènements sont trop proches, il devient impossible de les détecter, c'est le comportement dit "zeno".

C'est pourquoi, quand il y a moins de trois points entre chaque évènement, l'intégrateur arrête le calcul de la solution. Ensuite, le modèle pourrait être étendu en faisant rouler la balle jusqu'à son arrêt complet.

4.1.13 Contexte des balles molles

Considérons maintenant le fait que la balle soit déformée durant le contact avec le sol dans un contexte de balles molles. Le comportement de la balle durant le contact est illustré sur la Figure 4.13

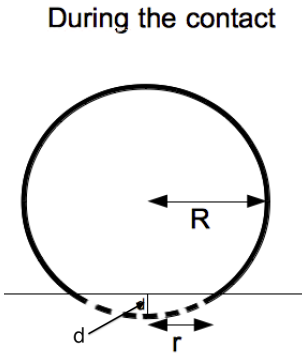


Figure 4.13 – Comportement de balle molle pendant le contact.

La partie qui rentre dans le sol et qui est en pointillé représente physiquement le fait que cette partie est écrasée lors du contact.

Vecteur des contraintes

Comme la balle n'est maintenant plus considérée comme un point mais comme un solide déformable, l'évènement a lieu quand le bord de la balle touche le sol. Cela signifie que le vecteur des contraintes prend maintenant en compte le rayon R de la balle :

$$\boldsymbol{c} = [y(t) - R].$$

Aussi longtemps que la balle ne touche pas le sol, les équations restent les mêmes, mais dès que le signe de c change, l'évènement est généré et le modèle est modifié pendant le contact entre la balle et le sol. La balle est déformée pendant le contact à cause des forces. La première force est répulsive et la seconde est dissipative [SDW95]. Cependant, comme le sol est supposé être un rigide solide, il applique sur la balle une force opposée par réaction. On suppose que la balle est déformée seulement dans le domaine élastique.

Expression de la force répulsive du sol sur la balle

En utilisant les résultats de l'élasticité linéaire, on a la force répulsive suivante $[m \cdot kg \cdot s^{-2}]$ en accord avec la loi de Hooke [Her09]

$$\boldsymbol{F}_{gr->ball} = -\sigma \; S \; \boldsymbol{u}_y$$
$$\sigma = E \; \epsilon$$

où σ est la contrainte $[kg \cdot m^{-1} \cdot s^{-2}]$, E le module d'Young $[kg \cdot m^{-1} \cdot s^{-2}]$, ϵ la déformation unidimensionnelle et S la surface de contact entre le sol et la balle.

Expression de S : Comme la surface de contact est un disque de rayon r, on a

$$S = \pi\ r^2.$$

Avec le théorème de Pythagore, on a

$$R^2 = r^2 + (R-d)^2 \quad \Longleftrightarrow \quad r^2 = 2Rd - d^2$$

En considérant que la déformation de la balle est très petite ou du premier ordre, on obtient $r = \sqrt{2R|d|}$. Finalement, il y a l'expression $S = 2\pi\ R\ |d|$.
.

Si le sol n'est pas plat (mais avec un rayon de courbure R_{ground}), R est pris comme la moyenne harmonique du rayon de courbure au contact

$$\frac{1}{R} = \frac{1}{R_{ground}} + \frac{1}{R_{ball}}.$$

Expression de σ : En considérant la loi de Hooke, on a $\sigma = E\ \epsilon$, où ϵ est la déformation de la sphère $\epsilon = \frac{\Delta V}{V_0} \approx -\frac{r}{R}$.

Finalement on obtient l'expression de la force répulsive :

$$\begin{aligned}\boldsymbol{F}_{gr->ball} &= 2\pi\ R\ |d| E \frac{\sqrt{2R|d|}}{R}\boldsymbol{u}_y \\ \boldsymbol{F}_{gr->ball} &= 2\pi\ E\sqrt{2R}|d|^{\frac{3}{2}}\boldsymbol{u}_y.\end{aligned}$$

Force dissipative

Si on considère une balle élastique, on a la force dissipative suivante [SDW95] :

$$\boldsymbol{F}_{diss} = -\mu\boldsymbol{v}.$$

Si on suppose que le matériau est viscoélastique au lieu d'élastique, on a la force suivante [SDW95]

$$\boldsymbol{F}_{viscodiss} = -\tilde{\mu}d^{1/2}\boldsymbol{v}.$$

On suppose maintenant que le matériau est élastique.

Système d'équations

Le système suivant est valide uniquement pendant le contact.

$$\begin{cases} m\dfrac{\mathrm{d}}{\mathrm{d}t}v_x = -\mu v_x, \\ m\dfrac{\mathrm{d}}{\mathrm{d}t}v_y = -m\ g + 2\pi\ E\sqrt{2R}|R - y(t)|^{\frac{3}{2}} - \mu v_y, \\ \dfrac{\mathrm{d}x}{\mathrm{d}t} = v_x, \qquad \dfrac{\mathrm{d}y}{\mathrm{d}t} = v_y, \\ v_x(t = t*) = v_{x^*}, \qquad v_y(t = T*) = v_{y^*}, \\ x(t = t*) = x^*, \qquad y(t = t*) = y^*, \\ c(t) = y(t) - R. \end{cases}$$

Dès que le signe de c passe de négatif à positif (la balle ne touche pas le sol), on change de modèle comme on l'a vu précédemment. Les conditions initiales sont la solution calculée

au dernier point.

Une alternative est de considérer la théorie d'impact de Hertz (voir [Lov27] et [Aba13]). La force résultante est alors donnée par une formulation différente mais le test utilise l'une ou l'autre formule de façon équivalente :

$$\boldsymbol{F}_{Hertz} = \frac{4}{3} E\sqrt{R}\ \ |R - y(t)|^{\frac{3}{2}}. \tag{4.19}$$

4.1.14 Résultats pour le contexte des sphères molles

Caractéristiques du modèle

Non linéaire	Type de nonlinéarité	Taille	Sparsité	Evènements
Oui	Ordre 2 par rapport à la vitesse	2	0	Oui

Table 4.6 – Caractéristiques du modèle.

Caractéristiques du solveur utilisé

Schéma	Ordre	Pas de temps	Tolérance	Parallélisme	Nombre de tâches
BDF	6	Variable	10^{-6}	Non	1

Table 4.7 – Solveur utillisé.

Comparons les courbes dans le cadre des sphères dures et dans le cadre des sphères molles. On trace la trajectoire du centre de gravité de la balle avec un rayon de $10cm$. Sur la Figure 4.14, les courbes semblent être les mêmes.

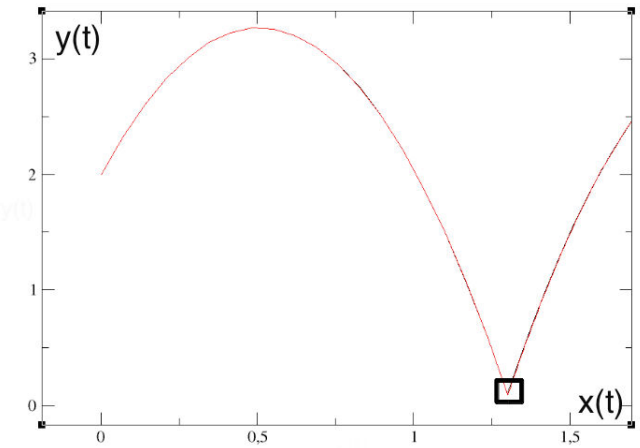


Figure 4.14 – Sphère molle (ligne rouge et continue) et sphère dure (ligne noire et en pointillé).

On donne le zoom sur le carré noir sur la Figure 4.15

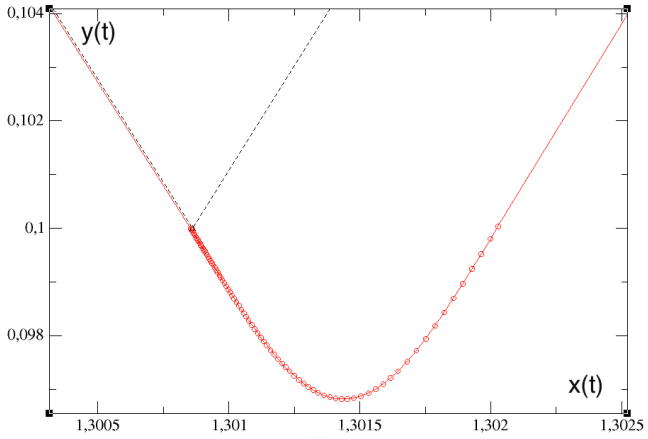


FIGURE 4.15 – Sphère molle (ligne rouge et continue) et sphère dure (ligne noire et en pointillé).

Le modèle utilisé pendant le contact a besoin de beaucoup de points. Comme on bascule entre deux modèles physiques, la trajectoire, dans le cade des sphères molles, est plus lisse que dans le contexte des sphères dures. Le temps de contact est de $1.196ms$, le pas de temps avant le contact est de l'ordre de $50ms$ et pendant le contact de quelques microsecondes.

4.1.15 Comparaison des résultats obtenus au moyen de deux solveurs

De même que précédemment on introduit la méthode de gestion des évènements dans un solveur dont les propriétés sont décrites dans le Tableau 4.7 et on compare avec les résultats donnés par d'autres solveurs implémentés dans OpenModelica, en particulier DASSL. Le challenge est de construire un modèle qui alterne entre deux types de systèmes d'équations en utilisant une condition de type if.

Code Modelica

Le modèle construit dans OpenModelica est

```
model SoftBouncingBall
  Real v_x(start = 1);
  Real v_y(start = 5);
  Real x(start = 0);
  Real y(start = 2);
  parameter Real Cx = 0.5;
  parameter Real rho = 1.293;
  parameter Real pi = 3.141592653;
  parameter Real S = pi * 0.1 * 0.1;
  parameter Real R = 0.1.
  parameter Real E=0.1*10^9;
  parameter Real density=500;
  parameter Real m = (4/3)*pi*R^3*density;
  constant Real g = 9.81;
equation
  if y > R then
    m * der(v_x) = -0.5 * Cx * rho * S *
                              sqrt(v_x ^ 2 + v_y ^ 2) * v_x;
    m * der(v_y) = -m * g - 0.5 * Cx * rho * S *
```

```
                                sqrt(v_x ^ 2 + v_y ^ 2) * v_y;
    der(x) = v_x;
    der(y) = v_y;
else
     m*der(v_x) = 0;
    m*der(v_y) = -m*g+2*3.14*E*sqrt(2*R)*abs(R-y)^(3/2);
     der(x) = v_x;
     der(y) = v_y;
end if;
end SoftBouncingBall;
```

Résultats

La variable y est tracée pour les deux solveurs sur la Figure 4.16.

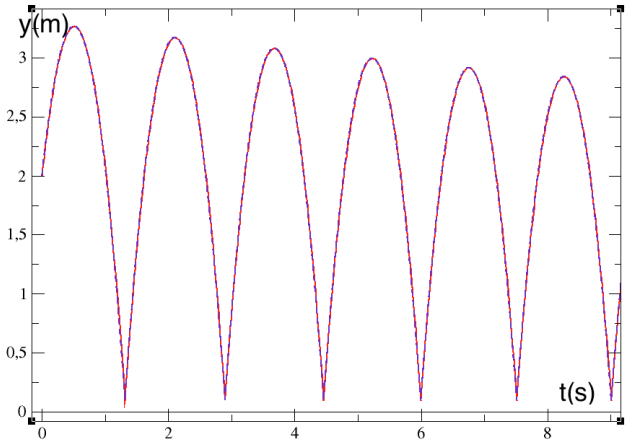


Figure 4.16 – Trajectoire du centre de gravité de la balle. La solution donnée par DASSL est en bleue et en ligne pointillée, la solution du solveur utilisé est en rouge et en ligne continue.

Comparons dans la Table 4.8 le temps CPU pour la simulation de la balle rebondissante pendant 10 secondes avec DASSL. Il apparait que DASSL prend plus de temps.

Temps CPU DASSL	Temps CPU solveur utilisé
0.061 s	0.045 s

Table 4.8 – Temps d'exécution

Un zoom du point de rebond à 7.51 s sur la Figure 4.17 affiche la différence entre les deux solutions qui restent très proches. Cependant cette observation ne concerne que le premier rebond et l'erreur se cumule d'un rebond au suivant. Cette erreur sur la détection de l'évènement n'est pas comprise dans l'erreur relative calculée par le solveur lors de l'adaptation du pas de temps sur la partie aérienne de la trajectoire (remise à zéro des conditions initiales à chaque détection d'évènement).

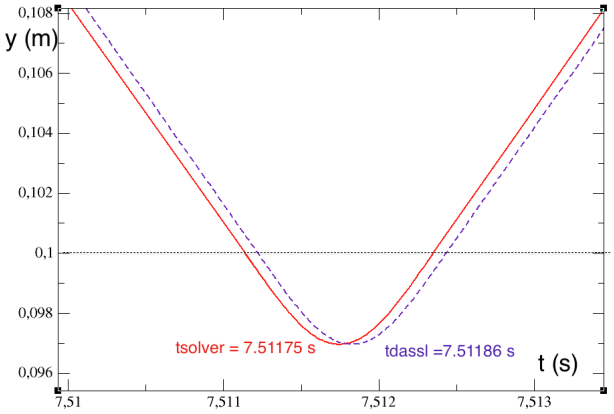
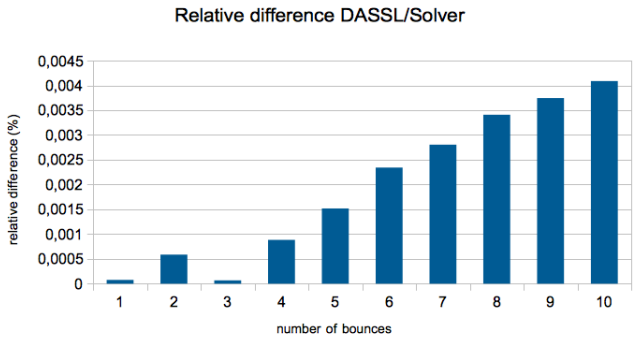
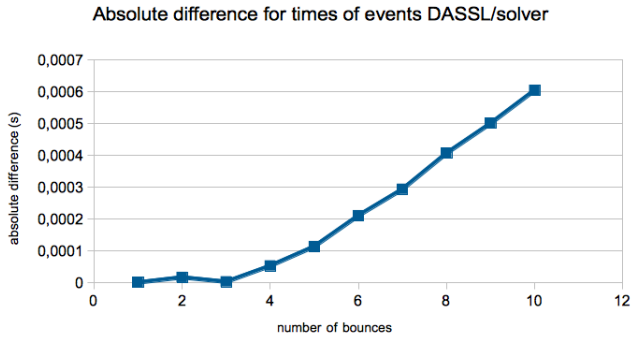


FIGURE 4.17 – Trajectoire temporelle du centre de gravité de la balle. DASSL en bleu et en ligne pointillée, le solveur utilisé en rouge et en ligne continue.

L'erreur relative sur les temps d'évènements entre DASSL et le solveur utilisé pour les dix premiers rebonds est tracée sur la figure suivante.



La différence absolue est également tracée



Pour le modèle de la balle molle la solution analytique n'est pas calculée et en conséquence la meilleure solution ne peut pas être déterminée.

En conclusion, la balle dans le contexte des sphères dures est un bon exemple de modèle hybride avec changement des conditions initiales, les équations restant les mêmes. La balle dans le contexte des sphères molles est un exemple de modèle hybride avec un changement

des équations phénoménologiques du modèle.

4.1.16 Modélisation avancée des évènements avec Modelica

Quand un évènement induit des changements dans les équations du modèle et dans la dimension du vecteur d'état, Modelica a actuellement des difficultés à gérer un tel challenge. Ce benchmark propose une technique en Modelica basée sur la condition if afin de prendre en compte les changements d'équations. Dans le contexte des sphères dures et molles, les variables d'état restent les mêmes pendant la modélisation. Mais au delà de ce benchmark, il existe également des modèles avec changement des variables d'état lié au changement d'équation. Pour implémenter de tels modèles au moyen d'OpenModelica toutes les variables d'état doivent être déclarées à l'initialisation. Cependant, seulement une partie d'entre elles est utilisée dans le modèle courant ; des équations triviales sont ajoutées au sous-système courant utile pour rendre égal le nombre d'équations et le nombre de variables d'état. Cependant, les équations d'état sont comprises dans le système même si elles n'évoluent pas dynamiquement.

4.1.17 Conclusion du cas de la balle rebondissante

Après avoir construit un solveur EDO avec gestion d'évènement, une question se pose : comment tester le solveur en termes d'évènement ? L'exemple de la balle rebondissante a été étudié pour y répondre.

Ce paragraphe montre comment l'exemple est intéressant pour tester la gestion d'évènements. Dans le contexte des sphères dures, la mise à jour du modèle consiste seulement en de nouvelles conditions initiales. Dans le contexte des sphères molles un changement drastique du modèle a lieu pour prendre en compte l'écrasement de la balle pendant le contact. Juste avant le contact le pas de temps utilisé pour simuler la trajectoire balistique aérienne est de l'ordre de $50ms$ tandis que pendant le contact le pas de temps est de quelques microsecondes. En conséquence l'exemple de la balle rebondissante fournit un modèle hybride raide pour tester un solveur EDO.

La solution analytique (sans considérer la résistance de l'air) de cet exemple permet d'avoir des mesures exactes tout en évitant des expérimentations. Quand la tolérance est suffisamment petite, la solution (avec une une résistance très faible de l'air) doit être très proche de la solution analytique et donc apporter une preuve de qualité par continuité.

L'exemple de la balle rebondissante dans le contexte des sphères molles et des sphères dures est une référence. Il peut être facilement utilisé pour tester d'autres solveurs EDO et apporter une comparaison avec la méthode de gestion de l'évènement présentée ici en termes de précision, de vitesse d'exécution et de stabilité.

Remarquons que ce modèle peut être compliqué à l'infini en lançant plusieurs balles. Il est possible de gérer les collisions aériennes entre les balles en utilisant le même type de contact défini pour le contact avec le sol.

Finalement, l'extension de ce benchmark à plusieurs balles pourrait ouvrir la possibilité de tester une approche multi-agents dans une vision grande taille.

4.2 Cas issu des équations de Saint-Venant

4.2.1 Présentation du problème

On considère le problème de l'écoulement gravitique superficiel intervenant dans de nombreuses situations telles que les prévisions météorologiques, la modélisation des océans,

les écoulements dans les rivières et dans les zones côtières, les débris dans les avalanches, etc.

On considère le mouvement d'une couche relativement fine de matière sous l'influence de la gravité le long d'un relief complexe, comme le fond d'un océan ou une montagne (voir Figure 4.18).

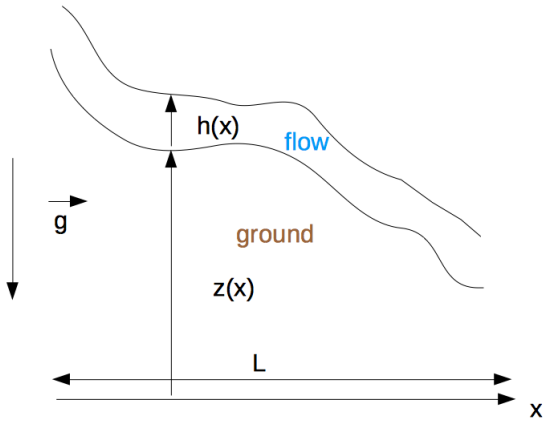


FIGURE 4.18 – Ecoulement d'eaux superficielles soumis à la gravité.

De tels écoulements en surface libre sont souvent modélisés par les équations d'eaux superficielles appelées également équations de Saint-Venant qui expriment la conservation du moment et de la masse. Les équations de Saint-Venant sont obtenues à partir des équations 3D incompressibles de Navier-Stokes en supposant la pression hydrostatique et en moyennant sur la direction verticale. Dans un espace à une dimension, la forme conservative est (voir [SV71]) :

$$\begin{aligned} &\frac{\partial \mathbf{h}}{\partial t} + \frac{\partial (\mathbf{h}u)}{\partial x} = 0 \\ &\frac{\partial (\mathbf{h}u)}{\partial t} + \frac{\partial}{\partial x}(\mathbf{h}u^2 + \frac{1}{2}g\mathbf{h}^2) + g\mathbf{h}\frac{\partial z}{\partial x} = -g\mathbf{h}S_f. \end{aligned} \tag{4.20}$$

Ici $\mathbf{h}$ est la hauteur de la matière, u est la vitesse dans la direction parallèle au lit de la rivière, g est la constante de gravité et S_f est le terme de friction estimé par la loi empirique de Darcy-Weishbach, $S_f = F\frac{u|u|}{8g\mathbf{h}}$. L'influence de la topographie s'exerce à travers la fonction $z(x)$ qui est la cote du relief.

Pour des raisons de simplicité, on suppose que $\dfrac{F}{8\mathbf{h}} = \lambda$ avec λ constant et on considère que $\left|\dfrac{\partial \mathbf{h}}{\partial x}\right|$ est très petit par rapport à $\left|\dfrac{\partial z}{\partial x}\right|$, et alors on néglige le terme $\dfrac{\partial}{\partial x}\left(\dfrac{\mathbf{h}^2}{2}\right)$ dans l'équation (4.20). Ainsi, la seconde équation de (4.20) devient

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2} + gz\right) = -\lambda|u|u. \tag{4.21}$$

On considère l'équation (4.21) sur $x \in [0,1]$ avec $z(x) = 0.1((1.4-x)^2 + \frac{0.2}{8}\sin(10\pi x))^2$ (voir Figure 4.19).

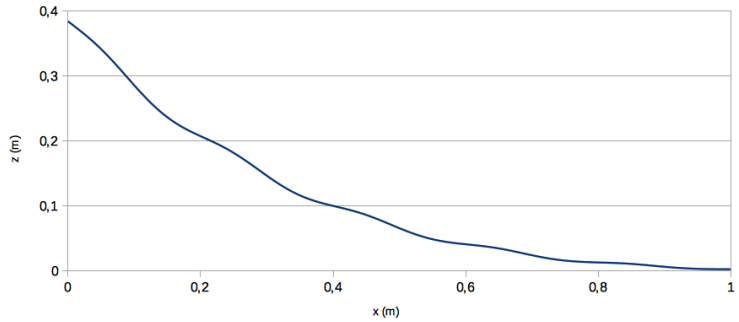


Figure 4.19 – Topographie du relief

.

4.2.2 Discrétisation des EDPs

Pour $N \in \mathbb{N}^*$, on introduit la subdivision $\{x_i\}_{0\leq i\leq N+1}$ de l'intervalle $[0,1]$ tel que $x_i = i\Delta x$, où $\Delta x = \frac{1}{N+1}$. On note également par $x_{i+\frac{1}{2}}$ le milieu de $[x_i, x_{i+1}]$, c'est-à-dire $x_{i+\frac{1}{2}} = \dfrac{x_i + x_{i+1}}{2}$. Après avoir utilisé la méthode des volumes finis pour la discrétisation spatiale de l'équation (4.21), le système suivant d'EDO défini sur l'intervalle de temps $[0,T]$ est obtenu : pour tout $1 \leq i \leq N$,

$$\frac{du_i}{dt} = -\frac{((u_i^2/2 + gz_i) - (u_{i-1}^2/2 + gz_{i-1}))}{\Delta x} - \lambda u_i|u_i|, \tag{4.22}$$

où z_i est l'approximation de la fonction $z := z(x)$ en $x = x_{i+\frac{1}{2}}$, u_i est l'approximation de la fonction $u := u(t,x)$ en $x = x_{i+\frac{1}{2}}$ et $u_{-1}(t) = u_{left}(t)$ est la condition limite de la vitesse. La raideur du modèle est approximativement égale à Δx et on a alors une non linéarité du second ordre.

4.2.3 Code Modelica

Une fois que la discrétisation a été effectuée, on peut modéliser le modèle de l'écoulement à l'aide du logiciel OpenModelica et du langage Modelica.

```
model SVtest
  import Modelica.SIunits;
  parameter Integer N = 10000;
  parameter SIunits.Length L = 1;
  parameter SIunits.Length dx = 0.0001;
  parameter Real lambda = 0.1;
  constant Real g = 9.81;
  parameter SIunits.Velocity ul = 0;
  SIunits.Velocity u[N](start = fill(0, N));
  parameter SIunits.Length z[N] = array(0.1 * ((1.4 - i * dx) * (1.4 - i * dx) + 0.2 / 8 * \\
  sin(10 * 3.14 * i * dx)) * ((1.4 - i * dx) * (1.4 - i * dx) + 0.2 / 8 * sin(10 * 3.14 * i * dx)) fo
  parameter SIunits.Length zl = 0.1 * ((1.4 - 0 * dx) * (1.4 - 0 * dx) + 0.2 / 8 *\\
   sin(10 * 3.14 * 0 * dx)) * ((1.4 - 0 * dx) * (1.4 - 0 * dx) + 0.2 / 8 * sin(10 * 3.14 * 0 * dx));
equation
  der(u[1]) = (-(u[1] * u[1] / 2 + g * z[1] - (ul * ul / 2 + g * zl)) / dx) - \\
  lambda * u[1] * abs(u[1]);
  for i in 2:N loop
    der(u[i]) = (-(u[i] * u[i] / 2 + g * z[i] - (u[i - 1] * u[i - 1] / 2 + g * z[i - 1])) / dx) - \\
    lambda * u[i] * abs(u[i]);
  end for;
end SVtest;
```

On peut ensuite exporter ce modèle sous forme d'un fichier .fmu.

4.2.4 Données numériques

Les tests ont été faits avec les données suivantes :

Paramètres	Symbole	Valeur
Taille d'un élément de longueur	Δx	0.0001
Nombre d'éléments de longueur	$N+1$	10000
Coefficient de Darcy-Weisbach	λ	0.1
Vitesse à la limite gauche	u_{left}	0
Gravité	g	9.81

TABLE 4.9 – Données numériques.

4.2.5 Résultats

Caractéristiques du modèle

Non linéaire	Type de nonlinéarité	Taille	Sparsité	Evènements
Oui	Ordre 2	10000	0.99990001	Non

Caractéristiques du solveur utilisé

Schéma	Ordre	Pas de temps	Tolérance	Parallélisme	Nombre de tâches
LIBDF	2	Fixe	Variable	Non	1

Résultats

Pour la discrétisation temporelle de l'équation (4.22), nous utilisons le schéma 2-LIBDF et le schéma 2-BDF afin de les comparer et de les vérifier. On utilise le schéma 2-BDF avec un très petit pas de temps $h = 10^{-5}$ pour construire la solution de référence. Ensuite, dans la Table 4.10, on mesure le temps CPU pour un code série sur Intel Core i5 2.5 Ghz. On utilise différents pas de temps h et on calcule l'erreur globale correspondante. Pour notre temps de simulation, on prend $T = 1$.

h	Temps CPU LIBDF (s)	Temps CPU BDF (s)	Ratio temps CPU time BDF/ LIBDF	Erreur globale LIBDF	Erreur globale BDF
$1/2$	0.012805	0.147817	11.54	0.970	0.257
$1/2^2$	0.019678	0.152786	7.76	0.234748	0.161628
$1/2^3$	0.057587	0.253571	4.40	0.07274	0.144567
$1/2^4$	0.067964	0.305854	4.50	0.014563	0.0297
$1/2^5$	0.079370	0.441204	5.55	0.000878	0.000371
$1/2^6$	0.149539	0.735913	4.82	0.0001	0.000127

TABLE 4.10 – Comparaison LIBDF BDF.

Sur la Figure 4.20, la vitesse est tracée à la fin de la simulation $T = 1$.

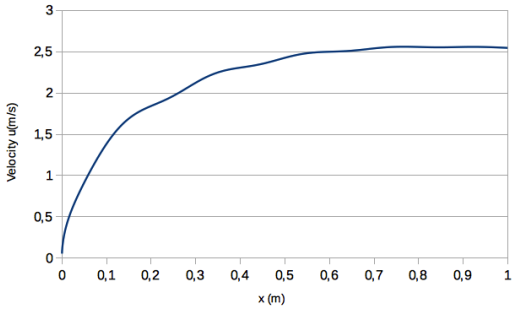


Figure 4.20 – Vitesse du fluide.

Conclusion

On remarque que le schéma LIBDF est toujours plus rapide que le schéma BDF (entre 4 et 11 fois plus rapide) pour approximativement la même erreur. On montre ainsi que le fait de ne pas résoudre un système non linéaire et d'évaluer la matrice Jacobienne pour chaque itération de la résolution de ce système permet d'augmenter de façon importante la rapidité d'exécution du système. Avec un pas de temps de $1/2^3$ et $1/2^4$ on remarque que notre solution est produite environ quatre fois plus rapidement tout en garantissant une erreur deux fois plus faible.

4.3 Cas d'un écoulement diphasique

4.3.1 Présentation du problème

On considère le problème de l'injection d'eau dans un milieu poreux saturé en huile avec des caractéristiques physiques homogènes. Quand l'eau est injectée dans le milieu poreux rempli d'huile, dans la direction verticale, dans une pièce cylindrique de matrice poreuse, entouré par une surface imperméable avec un fond également imperméable connecté avec le puit producteur, l'huile se déplace au niveau de l'interface commune $z = 0$ vers le bas. Des protubérances peuvent apparaître à cause de la viscosité de l'huile et de l'eau. L'eau s'injectera vers le bas à travers les cavités interconnectées et poussera l'huile vers le fond.

On note les quantités correspondantes aux phases aqueuse et huile par les indices w et o respectivement. La saturation S_α, est une quantité macroscopique qui décrit la fraction volumique moyenne ϕ_α , de la phase α dans un volume élémentaire représentatif (REV), normalisé par la porosité ϕ, c'est à dire :

$$S_\alpha = \frac{\phi_\alpha}{\phi}, \quad \alpha = w, o. \tag{4.23}$$

Cette définition implique la contrainte de saturation

$$S_w + S_o = 1. \tag{4.24}$$

L'équation de Buckley-Leverett modifié dérivant du modèle à écoulement diphasique (voir Chapitre 5 de [Che07], voir également [GNHH11]) est :

$$\phi \frac{\partial S_w}{\partial t} + \frac{\partial}{\partial z}\left(\frac{\lambda_w}{\lambda_w + \lambda_o}\boldsymbol{u}\right) + \frac{\partial}{\partial z}\left(\frac{\lambda_w \lambda_o}{\lambda_w + \lambda_o}\left(\frac{\partial}{\partial z} p_c(S_w) - (\rho_o - \rho_w)g\right)\right) = 0, \tag{4.25}$$

avec $\boldsymbol{u} \geq 0$ la vitesse d'écoulement de l'huile et de l'eau, ρ_α la densité de la phase $\alpha \in \{w, o\}$, λ_α le produit entre la mobilité η_α et la perméabilité intrinsèque k donné par,

$$\lambda_\alpha := k\eta_\alpha \quad \eta_\alpha := \frac{k_{r\alpha}(S_w)}{\mu_\alpha}, \tag{4.26}$$

avec μ_α la viscosité dynamique de la phase α et $k_{r\alpha}$ sa perméabilité relative. La pression capillaire p_c causée par la tension de surface à l'interface entre les deux phases est exprimée sur l'échelle macroscopique par la différence de pression entre les phases, $p_o - p_w = p_c$, où le modèle standard suppose que la pression capillaire, p_c, est une fonction de la saturation S_w seulement (voir par exemple la fonction de capillarité de Brooks-Corey [BC64]).

On introduit les fonctions d'une variable f et g définies par

$$f = \frac{\lambda_w}{\lambda_w + \lambda_o}, \quad g = \frac{\lambda_w \lambda_o}{\lambda_w + \lambda_o}.$$

L'équation (2.87) devient

$$\phi \frac{\partial S_w}{\partial t} + \frac{\partial}{\partial z}\left(f(S_w)\boldsymbol{u}\right) + \frac{\partial}{\partial z}\left(g(S_w)\left(\frac{\partial}{\partial z}p_c(S_w) - (\rho_o - \rho_w)g\right)\right) = 0. \tag{4.27}$$

4.3.2 Discrétisation des EDPs

On considère l'équation (4.27) sur $z \in [0, L]$. Pour $N \in \mathbb{N}^*$, on introduit la subdivision $\{z_i\}_{0 \leq i \leq N+1}$ de l'intervalle $[0, L]$ tel que $z_i = i\Delta z$, avec $\Delta z = \frac{L}{N+1}$. On note également $z_{i+\frac{1}{2}}$ le milieu de $[z_i, z_{i+1}]$ c'est-à-dire $z_{i+\frac{1}{2}} = \frac{z_i + z_{i+1}}{2}$. Après avoir utilisé la méthode des volumes fini pour la discrétisation spatiale de l'équation (4.27), le système d'EDO suivant, défini sur l'intervalle de temps $[0, T]$ est obtenu : pour tout $1 \leq i \leq N$,

$$\phi \frac{dS_{w,i}}{dt} + \frac{f(S_{w,i}) - f(S_{w,i-1})}{\Delta z}\boldsymbol{u} + \frac{g(S_{w,i+\frac{1}{2}})\Delta Pc_{i+\frac{1}{2}} - g(S_{w,i-\frac{1}{2}})\Delta Pc_{i-\frac{1}{2}}}{\Delta z} = 0, \tag{4.28}$$

avec $S_{w,i}$ une fonction du temps t approximant la fonction $S_w := S_w(z,t)$ en $z = z_{i+\frac{1}{2}}$, $\Delta Pc_{i+\frac{1}{2}} = \frac{p_c(S_{w,i+1}) - p_c(S_{w,i})}{\Delta z} - (\rho_o - \rho_w)g$ et $S_{w,i+\frac{1}{2}}$ est l'approximation de $S_w := S_w(z,t)$ en $z = z_{i+1}$ donnée par,

$$S_{w,i+\frac{1}{2}} = \begin{cases} S_{w,i} & \text{si } \Delta Pc_{i+\frac{1}{2}} > 0, \\ S_{w,i+1} & \text{sinon.} \end{cases} \tag{4.29}$$

4.3.3 Résultats

Caractéristiques du modèle

Non linéaire	Type de nonlinéarité	Taille	Sparsité	Evènements
Oui	Forte	1000	0.997002	Non

Caractéristiques du solveur utilisé

Schéma	Ordre	Pas de temps	Tolérance	Parallélisme	Nombre de tâches
LIBDF	2	Fixe	Variable	Non	1

Résultats

Pour la discrétisation temporelle de l'équation (4.28), on utilise notre schéma p-LIBDF et le schéma p-BDF afin de vérifier et de comparer les schémas entre eux. On utilise alors le schéma d'Euler pour construire notre solution de référence en prenant un pas de temps très petit. Pour notre test, on prend $N = 1000$. Pour les conditions aux limites, on prend $S_{w,-1} = 1$ et $S_{w,N+1} = 0$. Pour la condition initiale, on prend $S_{w,i} = 1$ pour $0 \leq i \leq 10$ et $S_{w,i} = 0$ pour $10 < i \leq N + 1$.
Pour la fonction de pression capillaire, on utilise

$$p_c = p_{ce} S_w^{-2},$$

et pour les perméabilités relatives, on utilise

$$kr_w(S_w) = S_w^2 \text{ et } kr_o(S_w) = (1 - S_w)^2.$$

Les tests ont été effectués avec les données suivantes : (voit Tableau 4.11)

Name	Symbol	Value	Unit
Grid Block size	Δz	0.01	m
Number of grid blocks	N	1000	
Depth of the reservoir	L	10	m
Sand porosity	ϕ	0.2	
Entry capillary pressure	p_{ce}	10^5	Pa
Sand permeability	k	10^{-12}	m^2
Water viscosity	μ_w	10^{-3}	$Pa.s$
Oil viscosity	μ_o	$5\ 10^{-3}$	$Pa.s$
Water density	ρ_w	1000	$kg.m^{-3}$
Oil density	ρ_o	800	$kg.m^{-3}$
Flow velocity	$\mathbf{u}$	0.1	$m.s$
Gravity	g	9.81	$m.s^{-2}$

Table 4.11 – Données numériques.

Sur la Figure 4.21, la saturation en eau est tracée pour différents temps de 0 à $10s$.

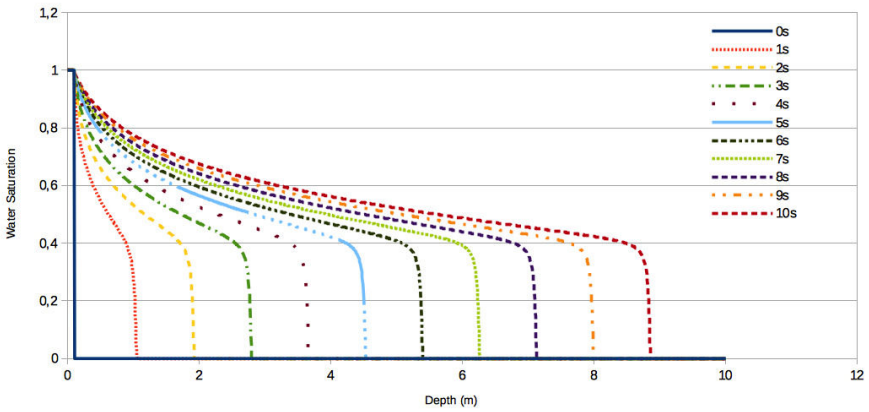


Figure 4.21 – Saturation en eau dans le réservoir pour différents temps.

Dans le Tableau 4.12, la comparaison entre le schéma 2-LIBDF et 2-BDF est donnée. On utilise un pas de temps fixe de $10^{-3}s$ pour les deux schémas et on vérifie l'erreur avec une solution de référence 2-BDF utilisant un pas de temps de $10^{-5}s$. On mesure le temps CPU pour un code séquentiel sur un processeur Intel Core i5 2.5 GHz.

Temps de Simulation (s)	Temps CPU LIBDF (s)	Temps CPU BDF (s)	Ratio temps CPU BDF/ LIBDF	Erreur globale LIBDF	Erreur globale BDF
1	0.13	1.12	8.62	0.00702	0.00699
2	0.24	2.25	9.38	0.012177	0.011996
3	0.37	3.41	9.22	0.017849	0.017572
4	0.47	4.64	9.87	0.026165	0.025875
5	0.57	5.73	10.05	0.034721	0.034267
6	0.71	6.89	9.70	0.037495	0.037046
7	0.94	8.15	8.67	0.048641	0.048007
8	0.95	9.38	9.87	0.050566	0.049956
9	1.11	10.46	9.42	0.06056	0.059187
10	1.19	11.52	9.68	0.064643	0.062012

TABLE 4.12 – Comparaison LIBDF et BDF.

Conclusion

On remarque que le schéma LIBDF est 8 à 10 fois plus rapide que le schéma BDF et ce pour une erreur globale équivalente. Ici, la résolution du système non linéaire utilisé par le schéma BDF est plus coûteuse que dans le cas des équations de Saint-Venant et ainsi le gain de temps du schéma LIBDF est plus significatif.

4.4 Cas linéaire de la diffusion de la chaleur dans une tige

Pour illustrer la méthode décrite dans la Section 3.2 du Chapitre 3, on utilise un exemple simple de système linéaire.

Soit une tige de métal de longueur L supposée assez fine pour permettre une modélisation 1D [Pat80]. Le phénomène est la conduction de la chaleur dans la tige où la température T à l'extrémité gauche est différente de celle à l'extrémité droite comme développé par M. Fourier [Fou22] et également [Pat80]. On s'intéresse à l'état transitoire aussi bien qu'à l'état permanent [Pat80] (voir également [Jan97]). L'équation aux dérivées partielles à laquelle obéit le phénomène de conduction de la chaleur dans la tige est un problème parabolique et est exprimée dans l'espace et le temps t au moyen de l'opérateur bien connu $div(grad)$.

$$\begin{cases} \dfrac{\partial T}{\partial t} = \dfrac{\partial}{\partial X}\left(\alpha \dfrac{\partial T}{\partial X}\right), \\ T(X, t=0) = T_0, \quad \forall X \in]0, L[, \\ T(X=0, t) = T_l, \quad \forall t \geq 0, \\ T(X=L, t) = T_r, \quad \forall t \geq 0. \end{cases} \tag{4.30}$$

avec $\alpha = \dfrac{\lambda}{\rho C_p} = \left[\dfrac{m^2}{s}\right]$, le coefficient de diffusivité thermique. Le champ initial de température est donné en $t = 0$ et les conditions aux bords de Dirichlet sont spécifiées aux

extrémités gauche et droite de la tige.

Si α est constant le long de la tige, l'équation devient alors $\dfrac{\partial T}{\partial t} = \alpha \Delta T$.

La solution analytique est décrite dans la Section 4.4.7 ; c'est une solution exacte du problème continu.
Ce modèle n'est pas restreint au phénomène de Fourier de conduction de la chaleur. En effet, l'opérateur $\nabla \cdot (\lambda \nabla)$ est un opérateur général qui modélise de nombreux phénomènes et est lié à l'équation de Laplace. Il exprime également la force visqueuse dans l'équation de Navier-Stokes $\nabla \cdot (\mu \nabla\, U)$ ([TZL]) où μ est la viscosité et U le vecteur vitesse. La loi de Darcy met en évidence le potentiel de flux $\frac{K}{\mu} \nabla\, p$ avec μ la perméabilité et p la pression [Dar56]. C'est pourquoi l'exemple proposé a une grande généralité.

4.4.1 Solution analytique

Il est facile de trouver une solution au problème (4.30). Ce point est développé dans la Section 4.4.7. La solution analytique est la référence pour comparer les résultats fournis par différents solveurs.

4.4.2 Construction des matrices A et B

Discrétisation spatiale d'ordre 2

Le modèle (4.30) contient une dérivée spatiale, donc comme décrit précédemment, il est possible de discrétiser le problème afin d'obtenir des matrices et des vecteurs de grande taille A et B. La discrétisation spatiale est obtenue au moyen d'un schéma d'ordre 2. Il y a Nx noeuds de taille dx et Nt pas de temps de taille dt. Le numéro des noeuds de la tige discrète est indiqué par l'indice i. Il vient que

$$\frac{\partial T(i)}{\partial t} = \frac{\alpha}{dx^2}(T(i+1) - 2T(i) + T(i-1)), \tag{4.31}$$

avec les conditions de Dirichlet $T(1,t) = T_l$, $T(L,t) = T_r$ et la condition initiale $T(X,0) = \dfrac{T_l + T_r}{2}$.
Pour le noeud numéro 2, le premier à être calculé, il vient

$$\frac{\partial T(2)}{\partial t} \approx \alpha \frac{\partial^2 T(2)}{\partial x^2} = \frac{\alpha}{dx^2}(T_l - 2T(2) + T(3)). \tag{4.32}$$

Pour le noeud numéro $n-1$, le dernier à être calculé, il vient que

$$\frac{\partial T(n-1)}{\partial t} \approx \alpha \frac{\partial^2 T(n-1)}{\partial x^2} = \frac{\alpha}{dx^2}(T_r - 2T(n-1) + T(n-2)). \tag{4.33}$$

Alors, le système d'EDO avec $i = 2, .., n-1$ approximant (4.30) s'écrit

$$\dot{T}(t) = AT(t) + B, \tag{4.34}$$

avec $A = \frac{\alpha}{dx^2} \begin{pmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 & -2 & 1 & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & -2 \end{pmatrix}$.

A est une matrice tridiagonale et $B = \frac{\alpha}{dx^2} \begin{pmatrix} T_l \\ 0 \\ \vdots \\ 0 \\ T_r \end{pmatrix}$.

Discrétisation spatiale à l'ordre 4

$\frac{\partial^2 T}{\partial x^2}$ est discrétisée au moyen d'un schéma centré d'ordre 4 ; à cause d'un effet de bord, le schéma n'est pas centré aux extrémités. L'expression générale de $\frac{\partial^2 T(i)}{\partial x^2}$ est

$$\frac{\partial^2 T(i)}{\partial x^2} \approx -\frac{1}{12}T(i-2) + \frac{4}{3}T(i-1) - \frac{5}{2}T(i) + \frac{4}{3}T(i+1) - \frac{1}{12}T(i+2). \tag{4.35}$$

A chaque extrémité, deux schémas décentrés d'ordre 4 sont utilisés et la matrice résolvante pentadiagonale est alors écrite pour $i = 3 \ldots n-2$.

Il est important de noter que le nombre de noeuds Nx peut être aussi large que voulu et par ce moyen, on peut fournir un système aussi large que souhaité. Cependant, ce système est linéaire et extrèmement creux comme il est pentadiagonal.
Grâce à la discrétisation spatiale plus ou moins fine, on peut produire des systèmes linéaires aussi raides que souhaités.

Construction des matrices à partir d'un outil de simulation orienté objet Modelica

A partir de l'outil Modelica il y a deux manières de construire la matrice A et le vecteur B :

– en lisant les équations dans le code

Le modèle de conduction de la chaleur peut également être construit au moyen de Modelica. La phénoménologie peut être introduite en éditant directement le code Modelica (voir le code ci-dessous)

```
model Heat_conduction_order2
  parameter Real T_left = 100;
  parameter Real T_right = 10;
  parameter Integer size = 10;
  parameter Real alpha = 0.001;
  parameter Real dx = 0.1;
  Real[size] T(start = array(55 for i in 1:size));
```

```
equation
  der(T[1]) = alpha / (dx * dx) * (-2 * T[1] + T[2] + T_left);
  for i in 2:size - 1 loop
  der(T[i]) = alpha / (dx * dx) * (T[i - 1] - 2 * T[i] + T[i + 1]);
  end for;
  der(T[size]) = alpha / (dx * dx) * (T[size - 1] - 2 * T[size] + T_right);
end Heat_conduction_order2;
```

Le modèle est très simple et il est facile de lire le code Modelica pour construire la matrice A et le vecteur B.

– en utilisant le FMU
La librairie Modelica fournit les éléments correspondants du bloc diagramme de la tige (voir Fig 4.22).

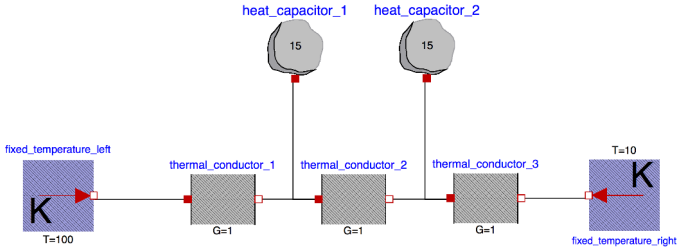


Figure 4.22 – Diagramme fonctionnel de conduction de la chaleur dans une tige en utilisant la librairie Modelica pour deux mailles. Les noeuds de la tige sont représentés par des condensateurs thermiques et chaque segment entre deux noeuds est représenté par un conducteur thermique.

Les équations du modèle ne sont pas écrites de façon évidente, la méthode décrite dans 3.2.1 est alors utilisée.

4.4.3 Complexification du modèle à cinq noeuds

On considère maintenant l'exemple de la conduction de la chaleur décrite précédemment avec cinq noeuds. La discrétisation au second ordre conduit au problème suivant écrit au moyen d'une matrice tridiagonale.

$$\dot{X}(t) = \frac{\alpha}{dx^2}\begin{pmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{pmatrix}, X(t) + \frac{\alpha}{dx^2}\begin{pmatrix} T_l \\ 0 \\ \vdots \\ 0 \\ T_r. \end{pmatrix} \tag{4.36}$$

Les cinq valeurs propres de A sont $\frac{\alpha}{dx^2}(-0.2679, -3.732, -2, -1, -3)$. La raideur (schéma d'ordre 2) est 13.93 ; il est évident que ce n'est pas un problème raide.

Notre but est de densifier cette matrice tandis que le problème conserve la même solution à des rotations près. Pour densifier la matrice, on utilise la méthode précédemment décrite des rotations successives. Alors le couplage, initialement limité aux coefficients des diagonales supérieures et inférieures, s'étendra à toutes les composantes au fur et à mesure que la matrice deviendra progressivement plus dense.

En pratique, on considère les quatres matrices de rotation d'angle $\frac{\pi}{3}$.

$$P_1^{\pi/3} = \begin{pmatrix} \cos(\frac{\pi}{3}) & -\sin(\frac{\pi}{3}) & 0 & 0 & 0 \\ \sin(\frac{\pi}{3}) & \cos(\frac{\pi}{3}) & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{4.37}$$

$$P_2^{\pi/3} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos(\frac{\pi}{3}) & -\sin(\frac{\pi}{3}) & 0 & 0 \\ 0 & \sin(\frac{\pi}{3}) & \cos(\frac{\pi}{3}) & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{4.38}$$

$$P_3^{\pi/3} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \cos(\frac{\pi}{3}) & -\sin(\frac{\pi}{3}) & 0 \\ 0 & 0 & \sin(\frac{\pi}{3}) & \cos(\frac{\pi}{3}) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{4.39}$$

$$P_4^{\pi/3} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \cos(\frac{\pi}{3}) & -\sin(\frac{\pi}{3}) \\ 0 & 0 & 0 & \sin(\frac{\pi}{3}) & \cos(\frac{\pi}{3}) \end{pmatrix}, \tag{4.40}$$

et les rotations sont alors successivement appliquées à A :

$$A_1 = (P_1^{\pi/3})^t \; A \; P_1^{\pi/3} = \frac{\alpha}{dx^2} \begin{pmatrix} -1.13 & -0.50 & 0.866 & 0 & 0 \\ -0.50 & -2.86 & 0.50 & 0 & 0 \\ 0.866 & 0.50 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{pmatrix},$$

$$A_2 = (P_2^{\pi/3})^t \; A_1 \; P_2^{\pi/3} = \frac{\alpha}{dx^2} \begin{pmatrix} -1.13 & 0.50 & 0.87 & 0 & 0 \\ 0.50 & -1.78 & 0.12 & 0.87 & 0 \\ 0.87 & 0.12 & -3.08 & 0.50 & 0 \\ 0 & 0.87 & 0.50 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{pmatrix},$$

$$A_3 = (P_3^{\pi/3})^t \; A_2 \; P_3^{\pi/3} = \frac{\alpha}{dx^2} \begin{pmatrix} -1.13 & 0.50 & 0.43 & -0.75 & 0 \\ 0.50 & -1.78 & 0.81 & 0.32 & 0 \\ 0.43 & 0.81 & -1.84 & 0.22 & 0.87 \\ -0.75 & 0.32 & 0.22 & -3.25 & 0.5 \\ 0 & 0 & 0.87 & 0.5 & -2 \end{pmatrix},$$

$$A_4 = (P_4^{\pi/3})^t \; A_2 \; P_4^{\pi/3} = \frac{\alpha}{dx^2} \begin{pmatrix} -1.13 & 0.50 & 0.43 & -0.37 & 0.65 \\ 0.50 & -1.78 & 0.81 & 0.16 & -0.28 \\ 0.43 & 0.81 & -1.84 & 0.86 & 0.24 \\ -0.37 & 0.16 & 0.86 & -1.88 & 0.29 \\ 0.65 & -0.28 & 0.24 & 0.29 & -3.37 \end{pmatrix}.$$

La sparsité évolue d'une matrice tridiagonale à une matrice complètement dense : successivement, on a les fractions d'éléments non nuls sur le nombre total d'éléments 13/25, 15/25, 17/25, 21/25, 25/25, montrant que la méthode proposée de densification marche (voir Figure 4.23). Il est facile de vérifier que les valeurs propres des matrices A_4, A_3, A_2, A_1 et A sont exactement les mêmes : $\frac{\alpha}{dx^2}(-0.2679, -3.732, -2, -1, -3)$. En conséquence, la raideur de cette famille de modèles reste la même. En excluant les angles triviaux comme $0, \pm\pi, \pm 2\pi$, ces matrices sont progressivement moins creuses d'une rotation à la suivante. Cependant, toujours en excluant les angles triviaux $0, \pm\pi, \pm 2\pi$, il est clair que $N-1$ rotations permettent d'obtenir une matrice complètement dense de taille $N \times N$ dans tous les cas.

Le calcul pour chaque problème devient de plus en plus lourd mais les solutions restent les mêmes à des rotations près. Cette méthode peut être appliquée aux modèles de très grande taille. Elle permet de comparer et d'améliorer les méthodes et les stratégies de parallélisation.

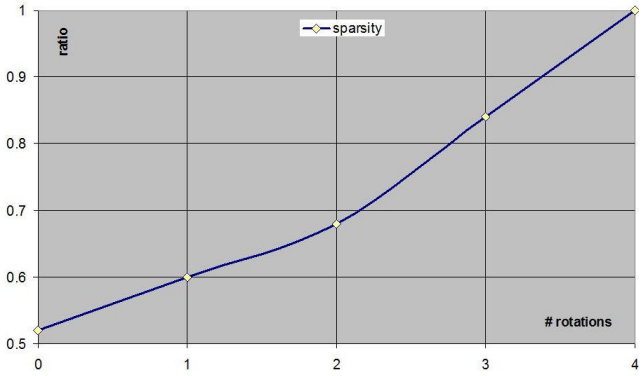


Figure 4.23 – Evolution du ratio du nombre de termes non nuls par rapport au nombre total de termes en fonction du nombre de rotations.

4.4.4 Résultats

Le modèle de la conduction de la chaleur est de taille adaptable et contient jusqu'à 10^6 termes non nuls dans un modèle avec 1000 mailles, soit une matrice 1000×1000. De plus, pour toutes les simulations, un contrôle a été effectué afin de vérifier que la solution produite est bien la solution exacte.

Caractéristiques du modèle

Non linéaire	Type de nonlinéarité	Taille	Sparsité	Evènements
Non	-	1000	Variable	Non

Caractéristiques du solveur utilisé

Schéma	Ordre	Pas de temps	Tolérance	Parallélisme	Nombre de tâches
BDF	Ordre 6	Variable	10^{-4}	Oui	1-8

Configuration du matériel de calcul

Les résultats suivants ont été obtenus en utilisant la machine suivante :
Intel® Xeon® Processor E5-2680 ; nombre de coeurs : 8 coeurs ; Clock Speed : 2,7 GHz ; Max Turbo Frequency : 3,5 GHz ; Cache : 20MB.

Résultats des simulations

Les résultats sont obtenus avec une tige de 1000 noeuds ce qui induit une matrice A de taille 1000×1000. A partir d'un schéma du deuxième ordre une matrice tridiagonale est obtenue ; ensuite rotation après rotation, la matrice A devient complètement dense. Les résultats des tests sont conservés chaque 100 rotations. Premièrement le nombre d'éléments non nuls est tracé en fonction du nombre de rotations sur la Figure 4.24

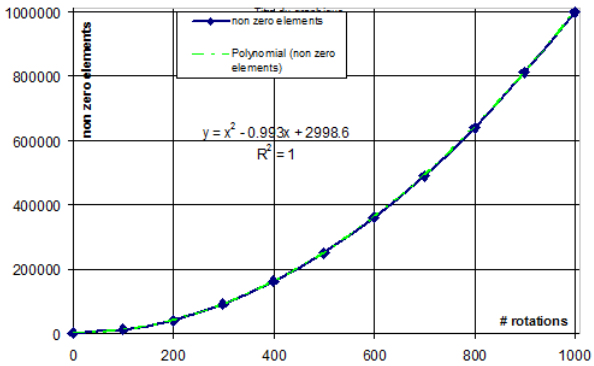


FIGURE 4.24 – Nombre de termes non nuls en fonction du nombre de rotations.

La courbe est parfaitement interpolée avec un polynôme d'ordre 2. Ainsi, le nombre d'éléments non nuls est proportionnel au carré du nombre de rotations (voir expression dans la Figure 4.24.)

On considère ensuite un solveur avec deux stratégies de parallélisation différentes en utilisant en particulier les directives OpenMP décrites dans la Section 3.3 du Chapitre 3 et on applique ces dix matrices de plus en plus denses au solveur et on regarde le temps de calcul du solveur utilisant les différentes matrices.

Stratégie de parallélisation 1 dite "bloc"

La matrice A est divisée en différentes sous-matrices. Le nombre de sous-matrices est égal au nombre de coeurs considérés. Par exemple, si on considère deux coeurs, un coeur travaille avec la première moitié de la matrice et le second coeur traite avec la seconde moitié. On dit alors que la parallélisation de la matrice A est par bloc.

Ensuite, le temps de calcul du solveur est tracé en fonction du nombre de termes non nuls sur la Figure 4.25. Quatre cas correspondant à un nombre différent de coeurs sont présentés. Les paramètres de la simulation de la conduction de la chaleur sont : A est une matrice $1000 \times 1000,$; $\alpha = 0.01SI$, $T_l = 373K$, $T_r = 273K$, $T_0 = 328K$, $dx = 0.01m$, $L = 10m$, $rtol = 0.0001$, $T_{end} = 3000\ s$.

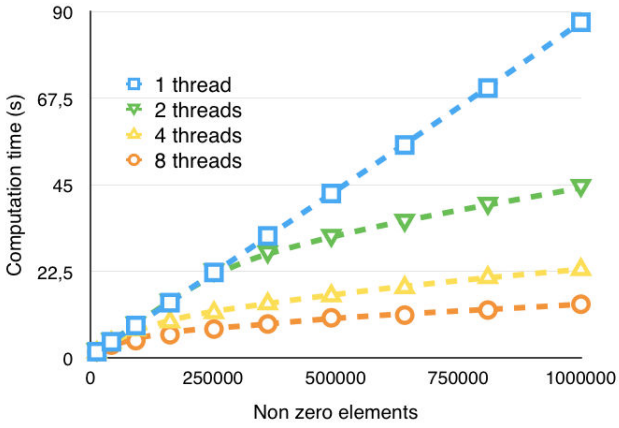


Figure 4.25 – Temps de calcul en fonction du nombre d'éléments non nuls pour la stratégie de parallélisation 1.

Et les speedups sont présentés sur la Figure 4.26

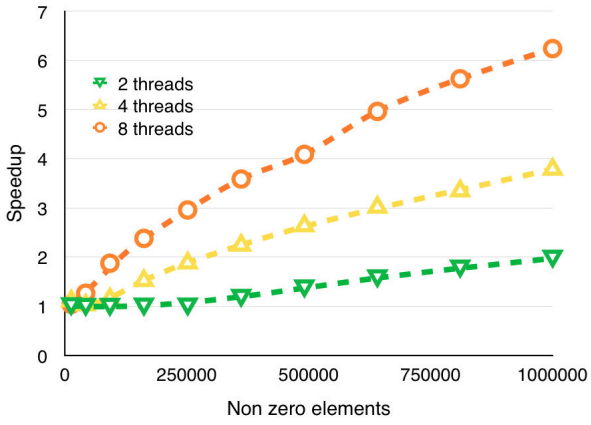


Figure 4.26 – Speedups du solveur 1.

Tout d'abord, le temps de calcul sur un coeur est linéaire. Ensuite, pour deux coeurs, on remarque un speedup pour un nombre de termes non nuls plus grand que 250000. Il en est de même pour quatre coeurs mais, dans ce cas, le speedup est observé pour 100000 termes non nuls. Cette limite est même plus petite pour 8 coeurs étant donné qu'on peut remarquer une accélération pour 50000 termes non nuls.

Stratégie de parallélisation 2 dite "par ligne"

La matrice A est divisée en différentes sous-matrices comme pour la stratégie de parallélisation 1 mais avec une manière différente. Le nombre de sous-matrices est toujours égal au nombre de coeurs considérés mais, par exemple, si deux coeurs sont considérés, la première ligne de la matrice est envoyé vers le premier coeur, ensuite la seconde ligne est envoyée vers le second coeur, la troisième vers le premier coeur à nouveau et ainsi de suite. Cela signifie que la parallélisation de la matrice A est dite en ligne. On garde les mêmes paramètres que pour la stratégie de parallélisation 1. Les résultats pour le temps de calcul et les speedups sont donnés dans la Figure 4.27 et la Figure 4.28.

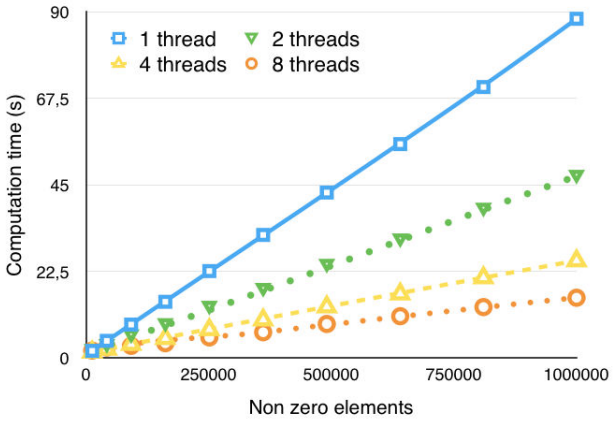


FIGURE 4.27 – Temps de calcul en fonction du nombre d'éléments non nuls pour la stratégie de parallélisation 2.

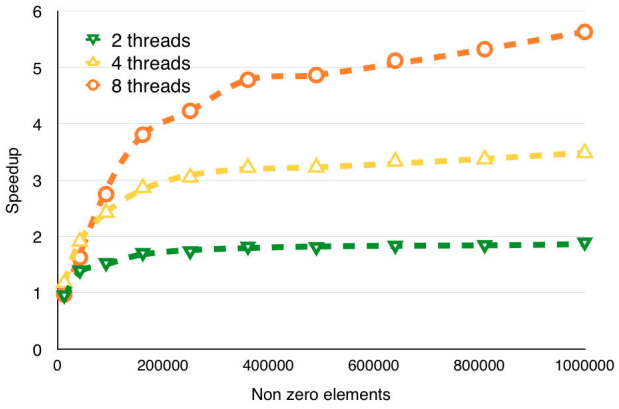


FIGURE 4.28 – Speedups de la stratégie parallélisation 2.

Tout d'abord, le temps de calcul pour un coeur est toujours linéaire. Ensuite, pour deux coeurs, quatre coeurs et huit coeurs, on remarque un speeedup pour un nombre d'éléments non nuls supérieurs à 3000.

On note deux principales différences entre les résultats des stratégies de parallélisation 1 et 2 :

– pour un coeur, la stratégie de parallélisation 2 est un peu plus lente.

– les speedups de la stratégie de parallélisation 2 sont plus élevés en particulier pour un faible nombre d'éléments non nuls.

L'explication du second point est simple : la stratégie de parallélisation 2 distribue ligne après ligne la matrice A au coeur 1, ce qui est coûteux. La stratégie de parallélisation 1 n'effectue pas cette opération car le bloc envoyé au coeur 1 correspond à la matrice A entière.

L'explication du second point est également simple : quand la matrice A est densifiée itérativement, les éléments non nuls apparaissent sur la première ligne, ensuite sur la seconde ligne et ainsi de suite. La matrice A est progressivement dense mais d'une manière hétérogène. Par exemple après 400 rotations la première moitié de la matrice A est beaucoup plus dense que la seconde moitié. Si deux coeurs sont considérés, la stratégie de la parallélisation 1 envoie la première moitié (la dense) au coeur 1 et la seconde moitié (la creuse) est envoyée au coeur 2. Le coeur 2 a donc un très petit problème au contraire du coeur 1. C'est pourquoi la parallélisation est inutile dans ce cas et que l'on n'observe pas de speedups pour les premières rotations. Comme la stratégie de parallélisation 2 distribue ligne par ligne aux différents coeurs, les sous-matrices sont aussi denses les unes que les autres et dans ce cas la parallélisation est efficace. On remarque dans ce cas une accélération même pour les premières rotations. Le benchmark met en évidence que la stratégie de parallélisation 1 n'est pas efficace pour des matrices hétérogènes. La stratégie de parallélisation 2 est plus efficace en dépit du fait qu'elle est plus coûteuse quand elle est utilisée avec un seul coeur. Ce benchmark valide la stratégie de parallélisation 2 mais pas la stratégie de parallélisation 1.

4.4.5 Implémentation d'une matrice de structure creuse

Afin de prendre en compte la sparsité d'une matrice qui contient beaucoup de zéros et d'éviter des calculs inutiles, il est nécessaire d'éviter la multiplication par zéro dans les calculs. C'est pourquoi une matrice sparse est codée avec une structure sparse nommée Compressed Sparse Row.

– Premièrement, un tableau est déclaré qui contient uniquement les éléments qui sont différents de zéro, ligne par ligne. Ce tableau est appelé *table_elements*. Sa taille est celle du nombre de termes non nuls.
– Un deuxième tableau contient l'index de colonne pour chaque élément de *table_elements* ; ce tableau est appelé *table_index_col*. Sa taille est la même que celle de *table_elements*.
– Un troisième tableau appelé *table_nze_row*, contient le nombre cumulé de termes non nuls pour chaque ligne. Sa taille est la même que le nombre de lignes de la matrice. Ces trois tableaux forment une structure qui contient uniquement les éléments non nuls de la matrice et leur position.

Il existe également une structure creuse CSC basée sur les colonnes au lieu des lignes avec trois tableaux : *table_elements*, *table_index_row*, *table_nze_col*.

Multiplication de structures creuses

On considère la matrice A structurée CSR et le vecteur $X(t)$. Il est facile de calculer la multiplication $AX(t)$ quand les élements de A sont ordonnés ligne après ligne. Mais maintenant, on considère la multiplication de A par une matrice de changement de base, $AP_i^{E_\alpha \to E_\beta} P_i^{E_\alpha \to E_\beta}$, qui doit être également structurée CSR. Le détail du calcul de cette multiplication est assez complexe. Pour rendre cette multiplication plus facile, on déclare $P_i^{E_\alpha \to E_\beta}$ en CSC.

On a donc implémenté une bibliothèque en C pour les opérations : matrice*matrice, vecteur*matrice, matrice*vecteur, avec la structure sparse appropriée.

Transposée d'une matrice creuse

Le calcul de $A_\alpha = (P_i^{E_\alpha \to E_\beta})^T \; A \; P_i^{E_\alpha \to E_\beta}$ nécessite la transposée de $P_i^{E_\alpha \to E_\beta}$.

Proposition 4.1. *Si la matrice P est déclarée comme CSR alors sa transposée a la même structure sparse mais décrite en CSC.*

En conséquence, il est facile de basculer d'une matrice creuse à sa transposée.

4.4.6 Valeurs propres de A

Proposition 4.2. *Les valeurs propres de A et A_α sont les mêmes.*

Démonstration. La preuve découle directement du fait que les matrices A et A_α sont semblables (voir Section 3.2.2 du Chapitre 3). □

4.4.7 Solution analytique

Fourier dans [Fou22] propose une méthode pour déterminer une solution analytique de (4.30). On considère la conduction de la chaleur (4.30) avec α constant. Il s'agit d'une équation aux dérivées partielles avec des conditions aux bords de Dirichlet non homogènes. Premièrement on transforme cette EDP en une équation parabolique avec des conditions aux bords homogènes en utilisant une nouvelle variable $u(t, x)$:

$$\begin{aligned} u(t,x) &= T(t,x) + \frac{T_l - T_r}{L}(x - L) - T_r, \\ u(0,x) &= T_0 + \frac{T_l - T_r}{L}(x - L) - T_r, \\ u(t,0) &= 0 \qquad\qquad u(t,L) = 0. \end{aligned}$$

D'après les méthodes usuelles de séparation de variables, la solution est le produit de deux fonctions indépendantes

$$u(t,x) = F(x)\, G(t). \tag{4.41}$$

Comme u est également une solution de (4.30), on a $\dfrac{G'(t)}{\alpha G(t)} = \dfrac{F''(x)}{F(x)}$. Deux solutions qui sont égales et qui ne dépendent pas des mêmes variables, sont constantes, et ici égales à $-k$

$$G'(t) = -k\alpha G(t) \qquad\qquad F''(x) = -kF(x).$$

On suppose que $k < 0$, il y a deux contraintes C_1 et C_2 telles que $F(x) = C_1 e^{x\sqrt{-k}} + C_2 e^{-x\sqrt{-k}}$.

Mais les conditions aux bords impliquent $F(0) = 0 = F(L)$, soit $C_1 = C_2 = 0$ et $T = -\frac{T_l - T_r}{L}(x - L) + T_r$.

On suppose maintenant que $k = 0$, il y a alors deux constantes C_1 et C_2 : $F(x) = C_1 x + C_2$. Mais avec les conditions aux limites, on a alors $F = 0$ et donc $T = -\dfrac{T_l - T_r}{L}(x - L) + T_r$. Seulement le cas $k > 0$ doit être étudié, dépendant de trois constantes C_1, C_2 et C_3

$$G(t) = C_1 e^{-k\alpha t} \qquad\qquad F(x) = C_2 \sin(\sqrt{k}x) + C_3 \cos(\sqrt{k}x).$$

Les conditions aux bords impliquent $C_3 = 0$ et il existe un entier positif n tel que $\sqrt{k} = n\frac{\pi}{L}$. Comme l'équation est linéaire, une combinaison linéaire des solutions est également solution.

$$u(t,x) = \sum_{n=1}^{+\infty} D_n \sin\left(\frac{n\pi x}{L}\right) e^{-\frac{n^2\pi^2\alpha t}{L^2}}. \tag{4.42}$$

La valeur de la condition initiale donne $u(x,0) = T_0 + \frac{T_l - T_r}{L}(x-L) - T_r$. Cette équation est un développement de Fourier qui donne les valeurs des coefficients

$$\begin{aligned} D_n &= \frac{2}{L}\int_0^L \left(T_0 + \frac{T_l - T_r}{L}(x-L) - T_r\right)\sin\left(\frac{n\pi x}{L}\right) dx \\ &= -2\frac{-T_0 n\pi + T_l n\pi + T_0\cos(n\pi)n\pi + T_r\sin(n\pi) - T_l\sin(n\pi) - T_r n\pi\cos(n\pi)}{n^2\pi^2}. \end{aligned}$$

Finalement on a la solution analytique suivante

$$T(t,x) = \sum_{n=1}^{+\infty}\left(D_n \sin\left(\frac{n\pi x}{L}\right) e^{-\frac{n^2\pi^2\alpha t}{L^2}}\right) - \frac{T_l - T_r}{L}(x-L) + T_r, \tag{4.43}$$

$$\tag{4.44}$$

avec $D_n = -2\frac{-T_0 n\pi + T_l n\pi + T_0\cos(n\pi)n\pi + T_r\sin(n\pi) - T_l\sin(n\pi) - T_r n\pi\cos(n\pi)}{n^2\pi^2}$.

Quand t tend vers l'infini le profil de température se stabilise comme $T(x) = -\frac{T_l - T_r}{L}(x-L) + T_r$ qui est une ligne droite.

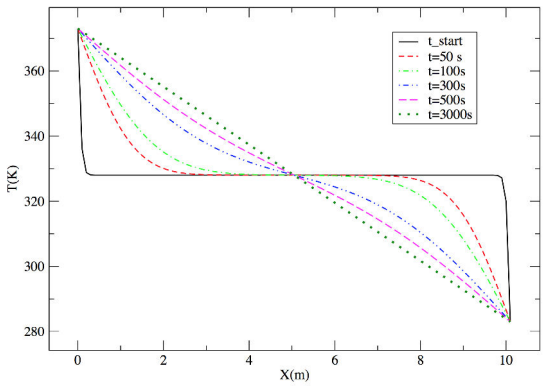


Figure 4.29 – Solution exacte du régime transitoire illustré par six courbes à différents instants indiqués dans la légende. La température initiale est la moyenne des températures fixées aux deux extrémités et la température finale stabilisée est une ligne droite de la température à l'extrémité gauche à la température à l'extrémité droite.

4.4.8 Conclusion

On traite de la modélisation et de la simulation de grands systèmes complexes et du test ainsi que de la comparaison de nouvelles approches parallèles ; le besoin d'un benchmark pour les modèles linéaires basés sur la matrice du système est clair.

Cette méthode utilise des techniques simples et sûres d'algèbre linéaire afin de produire une famille de benchmarks aussi complexes que nécessaires pour les solveurs EDOs. La

complexité d'un modèle de grande taille est ici associée au niveau du couplage entre les équations du modèle.

En partant d'un modèle de système industriel simulé par un solveur Modelica, on a tout d'abord exploré comment identifier la matrice désirée, à condition que le système soit linéaire, c'est-à-dire avec des coefficients constants. Cependant, ces modèles présentent des matrices creuses et partant de cette observation, une méthode de densification basée sur des changements de base a été mise au point et validée. L'exemple physique basé sur la diffusion thermique était adapté à nos besoins de benchmarks. La solution analytique de ce benchmark est disponible comme exemple que nous allons intégrer à OpenModelica.

Conclusion

On a montré dans cette thèse qu'il était possible d'accélérer la résolution des systèmes EDO de grande taille en utilisant plusieurs méthodes :

- en proposant de nouveaux schémas numériques pour le démarrrage démarrage puis pour le fonctionnement principal en évitant en particulier de résoudre un système non linéaire. Le schéma principal d'intégration, le schéma LIBDF, permet en effet de conserver les excellentes propriétés de stabilité du schéma BDF aussi bien en pas fixe qu'en pas variable tout en évitant l'utilisation d'un algorithme de Newton pour résoudre la partie implicite du schéma BDF. En effet cette partie implicite est linéarisée et cette linéarisation permet de diviser le temps de calcul jusqu'à 10 pour les problèmes fortement non linéaires comme un modèle d'écoulement diphasique qui tient compte de la pression capillaire. On a également pu montrer que dans le cas du schéma LIBDF, si le modèle possède des points d'équilibre stables, il est possible de majorer l'erreur globale par un terme qui ne dépend pas du temps total de simulation et qui n'explose pas avec la raideur. Le nouveau schéma LIBDF a fait l'objet d'un article qui est en révision pour le journal *Acta Applicandae Mathematicae* (voir [ABGS15]).
- en proposant une nouvelle méthode de gestion des discontinuités dans le cas des modèles hybrides. Le but de cette méthode est de déterminer rapidement les évènements en calculant en particulier le temps de l'évènement et la solution du système au moment de l'évènement considéré. Une fois ces valeurs déterminées, on doit rapidement redémarrer le solveur d'intégration. On a ainsi montré la rapidité de cette méthode sur un modèle de balle rebondissante et l'association de cette méthode avec un algorithme de démarrage rapide qui permet de simuler des modèles hybrides de manière rapide et efficace. La méthode de gestion des évènements a fait l'objet d'un article, également en révision, dans la revue *Journal of London Mathematical Society* (voir [GSB15b]).
- en effectuant une parallélisation efficace à la fois sur architecture en mémoire distribuée en utilisant les fonctions de la bibliothèque MPI et sur architecture en mémoire partagée en utilisant les directives OpenMP. Un profiling du solveur a tout d'abord été réalisé afin de distinguer les fonctions les plus coûteuses sur lesquelles il faut concentrer la parallélisation. On a également pu mettre en évidence l'importance du choix de la stratégie de parallélisation et de la répartition de la charge de travail sur les tâches de calcul. Une nouvelle méthode de construction de benchmark pour les solveurs parallèles a également été proposée dans cette thèse et cette méthode repose sur l'ajout progressif d'éléments non nuls dans la matrice d'un système linéaire initialement faiblement couplé. Cette méthode de construction de benchmark a fait l'objet d'une publication, également en révision, dans le journal *Publications de l'Institut Mathématique* (voir [GSB15a]).

Le nouveau schéma principal d'intégration a été testé jusqu'à l'ordre 3 et il serait intéressant de poursuivre l'utilisation de ce schéma jusqu'à l'ordre 6, ordre limite de zéro-stabilité. Pour cela il sera nécessaire d'utiliser une interpolation différente de l'interpolation polynomiale utilisée jusqu'à l'ordre 3. En effet le phénomène de Runge décrit dans le Chapitre 2 entraîne une erreur trop importante pour les ordres élevés. Il serait alors intéressant d'utiliser la méthode des splines ou les racines des polynômes de Tchebychev afin de garantir une erreur faible sur l'interpolation.

Le modèle de la balle rebondissante présenté afin de mettre en évidence la nouvelle méthode de gestion des évènements est de petite taille et il serait intéressant de construire un modèle de plus grande taille, avec de nombreux évènements, afin de tester la robustesse de la méthode proposée dans cette thèse. On pourrait envisager d'étudier un lancer de 1000 balles, qui rebondiraient les unes contres les autres.

On a également développé une méthode numérique, la Waveform Relaxation, qui permet de diviser la résolution du système complet en plusieurs sous-systèmes. On a vu que l'utilisation de la WFR nécessitait d'être dans des conditions bien particulières, en particulier il est nécessaire d'avoir des systèmes peu raides. De plus, on a également vu que l'utilisation de la WFR est mal aisée dans le cas du pas variable, ce qui empêche l'utilisation de la WFR dans le cadre d'un contrôle de l'erreur. La résolution des EDOs, en garantissant une erreur inférieure à une certaine tolérance, est pourtant très courante, et donc l'intérêt de la WFR est donc fortement diminuée. Dans la continuité de cette thèse, il serait intéressant de développer une méthode WFR en pas variable qui soit stable et rapide. Il serait également intéressant de reprendre les exemples de grande taille décrits dans le Chapitre 4 et d'utiliser une architecture multiprocesseurs de grande taile afin d'appliquer la méthode WFR à ces exemples.

De plus, l'intégration du schéma de démarrage, du schéma LIBDF et de la gestion des évènements est en cours dans le logiciel xMOD qui utilise le standard FMI. Cela permettra ainsi la résolution plus rapide de problèmes industriels complets et de grande taille, tels que des centrales EDF, pour lesquels l'utilisation des solveurs classiques est très coûteuse. A terme, on peut envisager l'intégration de ce solveur dans un outil tel qu'OpenModelica et promouvoir son utilisation dans les modèles utilisés dans le projet MODRIO [1] (voir [BBGBKS] et [SZB+]). L'accés à chacune des dérivées du modèle contenant dans le .fmu (voir Section 1.3.3 du Chapitre 1) doit également permettre une parallélisation aisée et l'utilisation de la WFR pour tous les modèles construits à partir du langage Modelica. Enfin, on peut envisager d'adapter ce solveur à l'étude de la variation de paramètres en implémentant ue option permettant de balayer automatiquement l'espace des paramètres. Ainsi, la vitesse d'exécution de ce solveur permettrait une étude plus rapide de l'espace des paramètres pour laquelle on a besoin de beaucoup de simulations.

1. Modrio est un projet européen ITEA3, impliquant de nombreux partenaires, dont IFPEN, plus d'informations sur : https ://itea3.org/project/modrio.html

Annexe A

Compléments sur les directives OpenMP

A.1 Motivations

CRAY et IBM avaient déjà leur propre jeu de directives avant 1997 mais la multiplication des machines multiprocesseurs à mémoire partagée a accéléré la création d'un standard. Le 28 octobre 1997, la plupart des industriels et des constructeurs ont adopté OpenMP (Open Multi Processing) comme un standard dit "industriel". Aujourd'hui, le seul organisme chargé de son évolution est l'ARB (Architecture Review Board). La dernière version 4.0 date d'octobre 2013.

A.2 Principes

Le développeur introduit lui-même les directives OpenMP dans le code. Dans les régions séquentielles seule la tâche maître exécute le programme jusqu'à une région parallèle où des processus sont créés et exécutent le travail à l'intérieur de cette région. A l a fin de cette région, les processus disparaissent et seule la tâche maître exécute la suite du programme.

La forme des directives OpenMP est la suivante :

```
sentinelle directive [clause[ clause] ...]
```

Cette ligne doit être ignorée lorsque l'option de compilation n'est pas spécifiée ; l'expression de la sentinelle dépend du langage utilisé. Le header qui définit le prototype de toutes les fonctions est OMP_LIB pour Fortran 95 et omp.h pour C/C++.

Par défaut, à l'intérieur d'une région parallèle, toutes les variables sont partagées et toutes les tâches concurrentes exécutent le même code. La synchronisation à la fin de la région parallèle est implicite et on ne peut pas faire de branchements de type "goto" vers l'intérieur ou l'extérieur d'une région parallèle.

A.3 Changement de statut des variables

Le statut par défaut de l'ensemble des variables est modifiable en utilisant la clause DEFAULT(PRIVATE). Toutes les variables ayant un statut privé sont indéfinies à l'entrée d'une région parallèle. Cependant, la clause FIRSTPRIVATE force l'initialisation des

variables privées à la dernière valeur qu'elles avaient avant la region parallèle. La clause LASTPRIVATE privatise une variable partagée à l'intérieur d'une région parallèle et permet de conserver la valeur calculée par la dernière tâche à la fin de la boucle.

Il est possible d'appeler des sous-programmes à l'intérieur des régions parallèles. Toutes les variables appelées dans le sous-programme sont par défaut privées.

Il est possible de préciser le nombre de tâches souhaitées à l'entrée d'une région parallèle avec la clause NUM_THREADS. Cette clause est identique au sous-programme OMP_SET_NUM_THREADS. Dans le cas où le nombre de tâches change entre différentes régions parallèles, il est nécessaire d'utiliser le sous-programme OMP_SET_DYNAMIC ou alors d'utiliser la variable d'environnement OMP_DYNAMIC égale à true lors de la compilation.

Il est possible d'imbriquer des régions parallèles mais il est alors nécessaire d'utiliser le sous-programme OMP_SET_NESTED ou la variable d'environnement OMP_NESTED à la valeur true.

A.4 Boucle parallèle

Dans la section 3.3 du Chapitre 3, on décrit la parallélisation des boucles principalement utilisée dans notre code. Il s'agit de répartir les itérations d'une boucle selon les différents coeurs de calcul. Seules les boucles for sont parallélisables, les boucles infinies de type while ne sont pas parallélisables. On répartit les itérations de la boucle selon différents modes grace à la clause SCHEDULE. Les indices d'itération sont tous privés. Si la clause NOWAIT est spécifiée à la fin de la boucle, il n'y a pas de synchronisation globale à la fin de celle-ci.

Différentes répartitions sont possibles et jouent un rôle important suivant la répartition du travail. Nous avons vu dans la section 4.4.4 du Chapitre 4 qu'un mauvais choix de répartition peut rendre une parallélisation inefficace. Les différents types de répartition sont :

- STATIC : les itérations sont réparties en paquets de taille donnée, l'ensemble des paquets est alors attribué de facon cyclique suivant l'ordre des tâches,
- DYNAMIC : les itérations sont réparties en paquets de taille donnée mais lorsqu'une tâche a fini son travail le paquet suivant lui est attribué,
- GUIDED : les itérations sont réparties en paquet dont la taille décroît exponentiellement ; lorsqu'une tâche a fini son travail, le paquet suivant lui est attribué.

La directive ORDERED exécute une boucle de façon ordonnée.

Dans le cas de variables partagées, il est possible de répartir la charge de travail sur les différentes tâches en utilisant les directives de réduction sur des opérations arithmétiques, logiques de fonctions intrinsèques. Des résultats partiels sont calculés par chaque tâche qui sont synchronisés pour déterminer le résultat final.

La directive PARALLEL DO est une fusion des directives PARALLEL et DO. La fin de cette directive, END PARALLEL DO, n'accepte pas de clause NOWAIT.

Des directives SECTION peuvent être définies à l'intérieur d'une construction SECTIONS. La portion de code qui suit la directive SECTION n'est exécutée que par une seule tâche. Le but de cette directive est de répartir l'exécution de différentes portions de codes indépendantes sur différentes tâches. On admet la clause NOWAIT pour empécher la synchronisation implicite à la fin de la section.

La construction MASTER exécute une portion de code par seulement la tâche maître.

Une fonction ou sous-programme appelé dans une région parallèle est exécuté par toutes les tâches.

A.5 Synchronisations

En fin de construction OpenMP, une barrière globale de synchronisation est implicitement appliquée. Cependant, on peut choisir explicitement une barrière avec la directive BARRIER. Cette directive permet de synchroniser l'ensemble des tâches concurrentes dans une région parallèle.

La directive ATOMIC permet de s'assurer qu'une variable partagée n'est lue et modifiée que par une seule tâche à la fois.

A.6 Mesures du temps

La fonction OMP_GET_WTIME permet de mesurer le temps de restitution. L'utilisation à deux endroits du code de cette fonction mesure le temps écoulé entre ces deux fonctions en secondes.

Annexe B

Complément sur les fonctions MPI

B.1 Environnement

Chaque programme qui contient des fonctions MPI doit inclure un header (en C/C++ le fichier mpi.h). L'appel au sous-programme MPI_INIT permet d'initialiser l'environnement nécessaire et MPI_FINALIZE désactive l'environnement.

Il est nécessaire également de définir un communicateur sur lequel vont porter les différentes opérations :MPI_COMM_WORLD.

Il est possible d'interrompre un programme MPI avant la fin de l'exécution en utilisant le sous-programme MPI_ABORT

On peut également connaître le nombre de processus dans un communicateur avec le sous-programme MPI_COMM_SIZE.

Le sous-programme MPI_COMM_RANK permet lui de connaitre le rang d'un processus.

Dans le cas où deux processus échangent un message, la communication est dite point à point. On définit alors le processus émetteur et le processus récepteur qui sont identifiés par leur rang. Le message est contenu dans une enveloppe qui est formée :

- du rang du processus émetteur,
- du rang du processus récepteur,
- de l'étiquette (tag) du message,
- du communicateur.

De plus, le type des données est défini dans le message.

L'opération d'envoi s'appelle MPI_SEND :

```
MPI_SEND(message,longueur,type, rang_dest,etiquette,comm,code)
```

Cette fonction envoie donc le contenu *message*, de taille *longueur*, de type *type*, étiqueté *etiquette*, au processus *rang_dest* dans le communicateur *comm*.

L'utilisation de cette fonction bloque l'exécution du programme tant que l'envoi n'est pas terminé.

De la même manière on définit la fonction de réception MPI_RECV :

```
MPI_RECV(message,longueur,type,rang_source,etiquette,comm,statut,code)
```

Cette fonction ne recevera un message de MPI_SEND que si ces deux fonctions ont le même *rang_source*, *rang_dest*, *etiquette* et *comm*. La variable *statut* contient des informations sur le *rang_source*, *etiquette* etc ...

De même que pour MPI_SEND, la fonction MPI_RECV bloque l'exécution du programme tant que la réception n'est pas terminée.

Les types MPI sont identiques aux types C/C++ mis à part le fait que l'on rajoute le préfixe MPI_ devant (MPI_DOUBLE par exemple).

Lorsqu'on ne veut pas spécifier le rang de l'émetteur pour la réception d'un message on peut écrire MPI_ANY_SOURCE.

On peut également combiner l'envoi et la réception d'un message avec la fonction MPI_SENDRECV :

```
MPI_SENDRECV(message_emis,longueur_message_emis,type_message_emis,rang_dest,
etiq_source,message_recu,longueur_message_recutype_message_recu,rang_source,
etiq_dest,comm,statu,code)
```

B.2 Communications collectives

Ce sont les communications qui concernent l'ensemble des processus du communicateur. Après une communication collective, une synchronisation globale est effectuée ; de plus les étiquettes ne sont jamais définies explicitement.

Les communications collectives ont trois types de sous-programmes :

- la synchronisation globale avec MPI_BARRIER
- les transferts de données :
  - envoi depuis un processus et réception de données par tous les processus : `MPI_BCAST(message,longueur,type,rang_source,comm,code)`,
  - envoi depuis un processus d'un message et réception d'une partie de ce message par tous les processus : `MPI_SCATTER(message_a_repartir,longueur_message_emis,type_message_emis,message_recu longueur_message_recu,type_message_recu,rang_source,comm,code)`,
  - envoi de parties de message par tous les processus et reception du message entier par un processus : `MPI_GATHER(message_emis,longueur_message_emis,type_message_emis,message_reçu, longueur_message_recu,type_message_recu,rang_dest,comm,code)`,
  - envoi de parties de message par tous les processus et reception du message entier par tous les processus correspondant à un MPI_GATHER suivi d'un MPI_BCAST : `MPI_ALLGATHER(message_emis,longueur_message_emis,type_message_emis,message_reçu, longueur_message_recu,type_message_recu,comm,code)`,
  - envoi sélectif de données par tous les processus. Le ième processus envoie la jème tranche au jème processus qui le place à l'emplacement de la ième tranche. `MPI_ALLTOALL(message_emis,longueur_message_emis,type_message_emis,message_reçu, longueur_message_recu,type_message_recu,comm,code)`
- opération de réduction avec ou sans diffusion du résultat MPI_ALLREDUCE et MPI_REDUCE() respectivement.
  `MPI_REDUCE(message_emis,message_recu,longueur,type,operation,rang_dest,comm,code)`
  `MPI_ALLREDUCE(message_emis,message_recu,longueur,type,operation,comm,code)`
  Dans le premier cas l'opération de réduction est faite sur *message_emis* réparti sur tous les processus et le processus *rang_dest* stocke le résultat dans *message_reçu*. Dans le second cas l'opération de réduction est identique mais le résultat *message_recu* est stocké sur l'ensemble des processus.

Lors des communications point à point, il existe deux modes : le mode synchrone et le mode asynchrone. Pour chaque mode, les appels peuvent être soit bloquants, soit non

bloquants.

Un appel bloquant est un appel qui suspend l'exécution du programme tant que l'envoi ou la réception n'est pas terminé. L'espace mémoire est réutilisé immédiatement après l'appel.

Un envoi synchrone oblige un processus qui envoie à attendre que le processus qui le reçoit soit prêt. L'exécution du programme par le processus émetteur est suspendue en attendant que le processus recepteur soit prêt.

Les avantages d'un envoi synchrone sont les suivants :
- peu d'espace mémoire consommé,
- rapide si le processus récepteur est prêt,
- la réception est assurée.

Les inconvénients sont :
- temps d'attente si le récepteur n'est pas prêt,
- risque de blocage.

Dans le cas où l'on souhaite éviter l'utilisation du mode synchrone, on utilise le mode dit bufferisé. Les données envoyées sont stockées dans un buffer en attendant d'être recues par un autre processus. Ces buffers doivent être gérés par l'utilisateur en utilisant MPI_BUFFER_ATTACH et MPI_BUFFER_DETACH. Pour l'envoi bufferisé, on utilise la commande :

```
MPI_BSEND(message,taille_message,type,rang_dest,etiquette,comm,code)
```

Les avantages sont les suivants :
- pas d'attente du processus récepteur,
- pas de blocage.

Les inconvénients sont les suivants :
- occupation mémoire supplémentaire,
- gestion manuelle des buffers et de leur taille,
- perte de temps si les récepteurs sont prêts lors de l'envoi,
- la réception n'est pas garantie,
- risque de mauvaise gestion de la taille des buffers.

Un compromis entre l'envoi synchrone et l'envoi bufferisé est l'envoi dit standard avec la commande MPI_SEND

```
MPI_SEND(message,taille_message,type,rang_dest,etiquette,comm,code)
```

Ce mode bascule automatiquement entre le mode bufferisé et le mode synchrone selon la taille des messages.

Les avantages de ce mode sont :
- le plus performant,
- le plus portable.

Il existe cependant des inconvénients :
- peu de contrôle sur le mode utilisé,
- risque de blocage si le mode synchrone est utilisé .

Les appels non bloquants permettent d'exécuter la communication en arrière-plan, on a alors un niveau de parallélisme supplémentaire. Pour ce type de communication, on utilise les commandes MPI_ISEND et MPI_IRECV.

Les avantages sont les suivants :
- gain de temps,
- pas de blocage.

Les inconvénients sont :
– surcoût,
– peu performant sur certaines machines.

Annexe C

Exemples de systèmes EDO de petite taille

C.1 Exemple linéaire peu raide : le sinus

Le premier système présenté dans cette annexe est le système très simple :

$$\begin{cases} \frac{\mathrm{d}y_1}{\mathrm{d}t} &= y_2, \\ \frac{\mathrm{d}y_2}{\mathrm{d}t} &= -y_1. \end{cases} \tag{C.1}$$

(C.1) est un système EDO linéaire de taille 2, couplé. La solution analytique de ce problème est connue :

$$y_1(t) = sin(t),$$
$$y_2(t) = cos(t).$$

La résolution de cette EDO avec un schéma BDF donne :

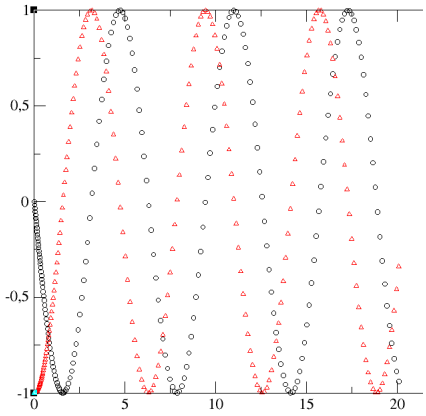


FIGURE C.1 – Solutions y_1 et y_2 en fonction du temps.

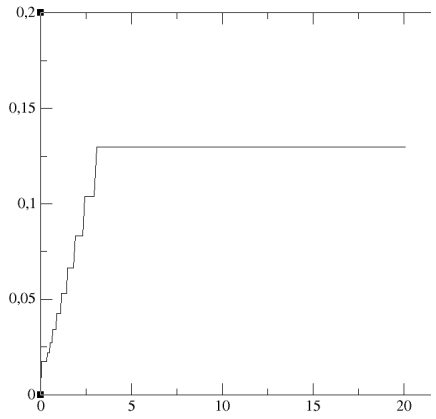


FIGURE C.2 – Evolution du pas de temps en fonction du temps.

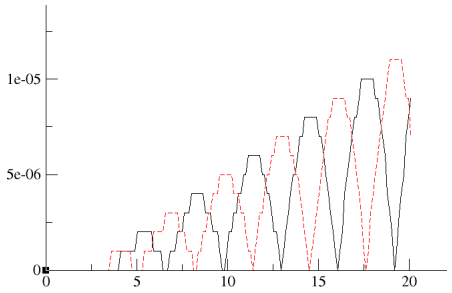


FIGURE C.3 – Valeur absolue de l'erreur absolue par rapport à la solution exacte.

C.2 Exemple non linéaire par rapport au temps, peu raide : battement avec enveloppe sinusoïdale

Le second modèle présenté dans cette annexe est un système EDO de taille 2 non linéaire par rapport au temps :

$$\begin{cases} \frac{\mathrm{d}y_1}{\mathrm{d}t} &= y_2, \\ \frac{\mathrm{d}y_2}{\mathrm{d}t} &= -\left(4\pi^2 + \left(\frac{4\pi^2}{10}\right)^2\right) y_1 + \frac{4\pi^2}{5}\sin(2\pi t)\sin(\frac{2\pi t}{10}). \end{cases} \tag{C.2}$$

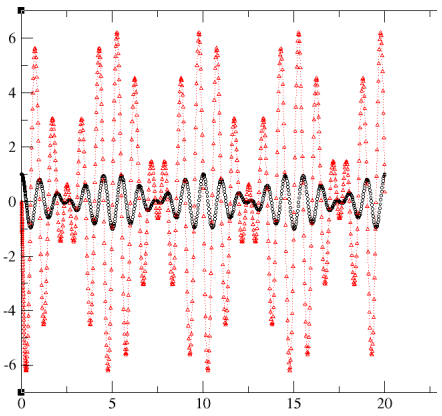


Figure C.4 – Solutions y_1 et y_2 en fonction du temps.

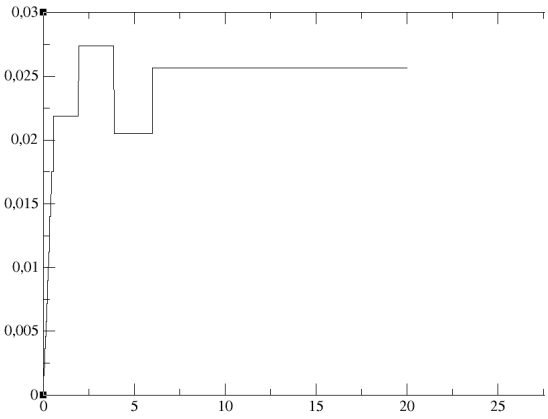


Figure C.5 – Evolution du pas de temps en fonction du temps.

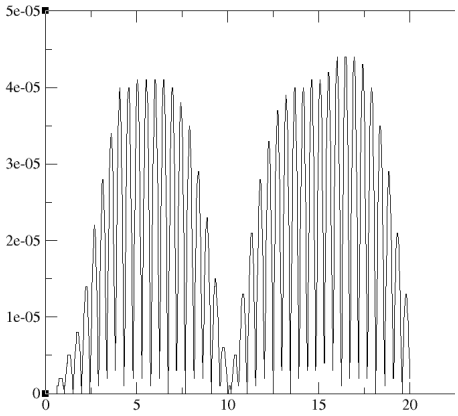


Figure C.6 – Valeur absolue de l'erreur absolue par rapport à la solution exacte.

C.3 Exemple non linéaire par rapport aux variables d'état, peu raide : équations de Lotka-Volterra

Le troisième modèle présenté ici est le modèle "proie-prédateur" :

$$\begin{cases} \frac{\mathrm{d}y_1}{\mathrm{d}t} &= \frac{3}{2}y_1 - y_1 y_2, \\ \frac{\mathrm{d}y_2}{\mathrm{d}t} &= (-3 + y_1)y_2. \end{cases} \tag{C.3}$$

(C.3) est un système EDO de taille 2 non linéaire par rapport aux variables d'état.

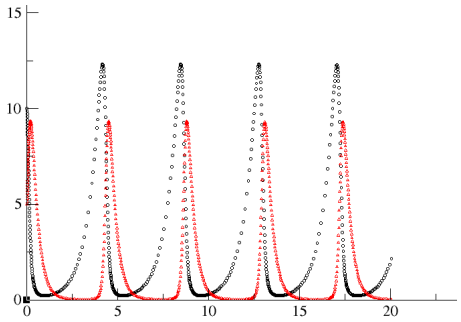


FIGURE C.7 – Solutions y_1 et y_2 en fonction du temps.

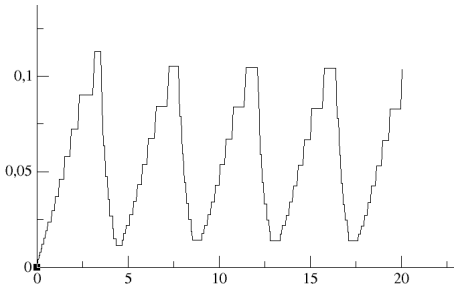


FIGURE C.8 – Evolution du pas de temps en fonction du temps.

Table des figures

Liste des tableaux

Bibliographie

[Aba13] Abaqus : Abaqus theory guide. *Simulia - 3.26.1 Discrete element analysis*, 2013.

[Abe07] O. Aberth : *Introduction to precise numerical methods.* Orlando, FL, USA, 2007.

[ABGS15] Léo Agélas, Jean Brac, Thibaut-Hugues Gallois et Thierry Soriano : A new robust numerical scheme for nonlinear ode systems. July 2015.

[AP98] Uri M Ascher et Linda R Petzold : *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.

[aRASV82] Lelarasmee andE., Ruehli, A.E. et A.L. Sangiovanni-Vincentelli : The waveform relaxation method for time-domain analysis of large scale integrated circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 1(3):131–145, Jul 1982.

[Bar] B. Barney : Introduction to parallel computing.

[BBGBKS] Jean Brac, Mongi Ben Gaïd, Abir Ben Khaled et Thierry Soriano : State-of-the-art methods of the parallel solutions of hybrid ode systems. Rapport technique, MODRIO.

[BC64] R. H. Brooks et A. T. Corey : *Hydraulic properties of porous media. Hydrology Papers 3.* 1964.

[BDP96] K. Burrage, C. Dyke et B. Pohl : On the performance of parallel waveform relaxations for differential systems. *Applied Numerical Mathematics*, 20:39–55, 1996.

[BF11] R.L. Burden et J.D. Faires : *Numerical Analysis.* 2011.

[BGCC+10] M. Ben Gaid, G. Corde, A. Chasse, B. Lety, R. De La Rubia et M. Ould Abdellahi : Heterogeneous model integration and virtual experimentation using xmod : Application to hybrid powertrain design and validation. *In 7th EUROSIM Congress on Modeling and Simulation*, Prague, Czech Republic, September 2010.

[BHJB77] G. D. Byrne, A. C. Hindmarsh, K. R. Jackson et H. G. Brown : A comparison of two ode codes : gear and episode. *Comput. Chemi. Eng.*, 1(2):133–147, 1977.

[BKEF14] Abir Ben Khaled-El Feki : *Distributed real-time simulation of numerical models : application to power-train.* Thèse de doctorat, Université de Grenoble, 2014.

[BM08] V.A. Botta et Meneguette JR. M. : Concerning the stability of bdf methods. *In International Conference of Numerical Analysis and Applied Mathematics*, volume 1048, pages 64–67. Kos, AIP Conference Proceedings, 2008.

[Bra02] Rebecca M. Brannon : Rotation : A review of useful theorems involving proper orthogonal matrices referenced to three-dimensional physical space. Rapport technique, Sandia National Laboratories, Albuquerque, NM, USA, 2002.

[But63] J.C. Butcher : Coefficients for the study of runge-kutta integration processes. *Journal of the Australian Mathematical Society*, 3(2):185–201, may 1963.

[CGG90] M. Caivo, T. Grande et R. D. Grigorieff : On the zero stability of the variable order variable stepsize bdf-formulas. *Numer. Math.*, 57:39–50, 1990.

[Che07] Z. Chen : *Reservoir Simulation : Mathematical Techniques in Oil Recovery*. PA, USA, 2007.

[CL55] E. A. Coddington et N. Levinson : *Theory of Ordinary Differential Equations*. New-York, 1955.

[CM93] M. Calvo et L. Montijano, J.I. an Rández : A_0-stability of variable stepsize bdf methods. *Journal of Computational and Applied Mathematics*, 45:29–39, 1993.

[Dah63] G.G. Dahlquist : A special stability problem for linear multistep methods. *BIT*, 3:27–43, 1963.

[Dar56] Henry Darcy : *Les Fontaines Publiques de la Ville de Dijon*. Victor Dalmont, Paris, France, 1856.

[EB12] Steven D. Eppinger et Tyson R. Browning : *Design Structure Matrix Methods and Applications*. MIT Press, 2012.

[EM97] H. Elmqvist et S. E. Mattsson : An introduction to the physical modeling language modelica. *In 9th European Simulation Symposium*, Passau, Germany, October 1997.

[Fou22] M. Fourier : *Théorie Analytique de la Chaleur*. Firmin Didot, Père et Fils, Paris, France, 1822.

[Fri03] P. A. Fritzson : *Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2003.

[Gau97] W. Gautschi : *Numerical Analysis : an Introduction,*. Boston, MA, 1997.

[Gea71] C.W. Gear : *Numerical initial value problems in ordinary differential equations*. 1971.

[GLS96] W. Gropp, E. Lusk et A. Skjellum : A high-performance portable implementation of the mpi message passing interface. *Parallel Computing*, 1996.

[GNHH11] M.J. Golding, J.A. Neufeld, M.A. Hessea et H.E. Huppert : Two-phase gravity currents in porous media. *J. Fluid Mech.*, 678:248–270, 2011.

[Gri83] R.D. Grigorieff : Stability of multistep methods on variable grids. *Numer. Math.*, 42:359–377, 1983.

[GSB15a] Thibaut-Hugues Gallois, Thierry Soriano et Jean Brac : Benchmark of adjustable size and coupling to test parallelization strategies of odes solvers. June 2015.

[GSB15b] Thibaut-Hugues Gallois, Thierry Soriano et Jean Brac : A new event handling solver for ode hybrid systems compared to modelica dassl. May 2015.

[HA08] Hassen HADJ AMOR : *Contribution au prototypage virtuel de systèmes mécatroniques basé sur une architecture distribuée HLA. Expérimentation sous les environnements OpenModelica-OpenMASK.* Thèse de doctorat, Université du Sud Toulon Var, 2008.

[HC90] S. Y. R. HUI et C. CHRISTOPOULOS : Numerical simulation of power circuits using transmission-line modelling. *In IEE proceedings, Part A. Physical science, Measurements and instrumentation, Management and Education, Reviews*, volume 6, pages 379–384, 1990.

[Hen62] P. HENRICI : *Discrete Variable Methods in Ordinary Differential Equations.* New York, 1962.

[Her09] J. HERNDON : Young's modulus. Rapport technique, Danville Area Community College, 2009.

[HN93] E. HAIRER et S.P. NOERSETT : *Solving Ordinary Differential Equations II : Stiff and Differential-Algebraic Problems.* 1993.

[HW83] E. HAIRER et G. WANNER : On the instability of the bdf formulas. *Siam J. Numer. Anal.*, 20(6), December 1983.

[IHAR09] J. IBANEZ, V. HERNANDEZ, E. ARIAS et P. A. RUIZ : Solving initial value problems for ordinary differential equations by two approaches : Bdf and piecewise-linearized methods. *Comp. Phy. Com.*, 180(5):712–723, 2009.

[Ise96] A. ISERLES : *A first course in the Numerical Analysis of Differential Equations.* 1996.

[Jan97] Jan JANSSEN : *Acceleration of Waveform Relaxation Methods for Linear Ordinary and Partial DIfferential Equations.* Thèse de doctorat, Katholieke Universiteit Leuven, Celestijnenlaan 200A - B-3001 Heverlee, Belgium, December 1997.

[JSC01] Lygeros J., Sastry S. et Tomlin C. : The art of hybrid systems. July 2001.

[JSD80] K.R. JACKSON et R. SACKS-DAVIS : An alternative implementation of variable step-size multistep formulas for stiff odes. *ACM Trans. Math. Software*, 6:295–318, 1980.

[KA01] B. KISAČANIN et G. C. AGARWAL : *Linear control systems : with solved problems and Matlab examples (University Series in Mathematics).* 2001.

[Kut01] Martin Wilhelm KUTTA : Beitrag zur naehrungsweisen integration totaler differentialgleichungen. *Zeitschrift fuer Mathematik und Physik*, 46:435–463, 1901.

[Lag77] J.L. LAGRANGE : Leçons élémentaires sur les mathématiques, données à l'ecole normale en 1975. *In Oeuvres VII*, pages 183–287. Gauthier-Villars, 1877.

[LFF10] Liu LIU, Felix FELGNER et Georg FREY : Comparison of 4 numerical solvers for stiff and hybrid systems simulation. *In Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8. IEEE, 2010.

[LLK+99] J. LIU, X. LIU, T-K. J. KOO, B. SINOPOLI, S. SASTRY et E. A. LEE : A hierarchical hybrid system model and its simulation. *In Proceedings of the 38th IEEE conference on Decision and Control*, volume 4, pages 3508–3513, December 1999.

[Lov27] A.E.H. LOVE : A treatise on the mathematical theory of elasticity. *Cambridge University Press*, 1927.

[LST01] J. Lygeros, S. Sastry et C. Tomlin : The art of hybrid systems. July 2001.

[M̈74] M et al Mäkelä : On the concepts of convergence, consistency, and stability in connection with some numerical methods. *Numer. Math.*, 22:261–274, 1974.

[Mar66] M. Marden : Geometry of polynomials. *Amer. Math. Soc.*, 1966.

[Mos99] P.J. Mosterman : An overview of hybrid simulation phenomena and their support by simulation packages. *In Hybrid Systems Computation and Control, lectures notes in Computer Science*, volume 1569, pages 165–177, 1999.

[NG97] U. Nowak et S. Gebauer : *A new Test Frame for Ordinary Differential Equation Solvers*. 1997.

[Pat80] Suhas V. Patankar : *Numerical Heat Transfer and Fluid Flow*. Hemisphere Publishing Corporation, 1980.

[Pet82] L. R. Petzold : A description of dassl : A differential/algebraic system solver. 1982.

[R.79] Jeltsch R. : a_0-stability and stiff stability of brown's multistep multiderivative methods. *Numer. Math.*, 32:167–181, 1979.

[Sch91] W. E. Schiesser : *The numerical method of lines : Integration of partial differential equations*. Acamedic Press (San Diego), 1991.

[SD80] R. Sack-Davis : Fixed leading coefficient implementation of sdformulas for stiff odes. *ACM Transactions on Mathematical Software*, 6(4):540–562, 1980.

[SDW95] J. Schäfer, S. Dippel et D.E. Wolf : Force schemes in simulations of granular material. *Les Editions de Physique*, 1995.

[Ske86] R.D. Skeel : Construction of variable stepsize multistep formulas. *Math. Comp.*, 47:503–510, 1986.

[SM03] E. Sueli et D. Mayers : *An Introduction to Numerical Analysis*. 2003.

[Smi06] Simon J. Smith : Lebesgue constants in polynomial interpolation. *Annales Mathematicae et Informaticae*, 33:109–123, 2006.

[SV71] A. J. C. Saint-Venant : Théorie du mouvement non permanent des eaux, avec application aux crues des rivières et à l'introduction de marées dans leurs lits. *C. R. Acad. Sc. Paris*, 73:147–154, 1871.

[SVVL+97] A. Sandu, J.G. Verwer, M. Van Loon, G.R. Carmichael, F.A. Potra, D Dabdub et J.H. Seinfeld : Benchmarking stiff ode solvers for atmospheric chemistry problems- i. implicit vs explicit. *Atmospheric Environment*, 31(19): 3151–3166, 1997.

[SZB+] Thierry Soriano, Manel Zerelli, Jean Brac, Thibaut-Hugues Gallois et Mongi Ben Gaïd : State-of-the-art analysis of complex systems with varying parameters. Rapport technique, MODRIO.

[TvdW99] P. Timmerman et J. P. van der Weele : On the rise and fall of a ball with linear or quadratic drag. *American Association of Physics Teachers*, 1999.

[TZL] C.H. Tai, Y. Zhao et K.M. Liew : Parallel multigrid computation of unsteady incompressible viscous flows using a matrix-free implicit method and high-resolution characteristic-based scheme. *Computer Methods in Applied Mechanics and Engineering*.

[ZYM08] Fu Zhang, Murali Yeddanapudi et Pieter Mosterman : Zero-crossing location and detection algorithms for hybrid system simulation. *17th IFAC world congress - XCOEX South Korea*, 17-pert I:7967–7972, 2008.

université PARIS-SACLAY
ÉCOLE DOCTORALE
INTERFACES
Approches Interdisciplinaires : Fondements, Applications et Innovation

Titre : Amélioration de la rapidité d'exécution des systèmes EDO de grande taille issus de Modelica

Mots clés : edo, modelica, grande taille, accélération, résolution

Résumé : L'étude des systèmes aux équations différentielles ordinaires vise à prédire le futur des systèmes considérés. La connaissance de l'évolution dans le temps de toutes les variables d' état du modèle permet de prédire de possibles changements radicaux des variables ou des défaillances, par exemple, un moteur peut exploser, un pont peut s'écrouler, une voiture peut se mettre à consommer plus d'essence. De plus, les systèmes dynamiques peuvent contenir des dérivées spatiales et leur discrétisation peut ajouter un très grand nombre d'équations.
La résolution des équations différentielles ordinaires est alors une étape essentielle dans la construction des systèmes physiques en terme de dimensionnement et de faisabilité. Le solveur de tels systèmes EDOs doit être rapide, précis et pertinent.

En pratique, il n'est pas possible de trouver une fonction continue qui soit solution exacte du problème EDO.
C'est pourquoi, des méthodes numériques sont utilisées afin de donner des solutions discrèes qui approchent la solution continue avec une erreur contrôlable. La gestion précise de ce contrôle est très importante afin d'obtenir une solution pertinente en un temps raisonnable.

Cette thèse développe un nouveau solveur qui utilise plusieurs méthodes d'amélioration de la vitesse d'exécution des systèmes EDOs. La première méthode est l'utilisation d'un nouveau schéma numérique. Le but est de minimiser le coût de l'intégration en produisant une erreur qui soit le plus proche possible de la tolérance maximale permise par l'utilisateur du solveur. Une autre méthode pour améliorer la vitesse d'exécution est de paralléliser le solveur EDO en utilisant une architecture multicoeur et multiprocesseur. Enfin, le solveur a été testé avec différentes applications d'OpenModelica.

Title : Improvement of execution speed of large scale ODE systems from Modelica

Keywords : edo, large scale, speed, modelica

Abstract : The study of systems of Ordinary Differential Equations aims at predicting the future of the considered systems. The access to the evolution of all states of a system's model allows us to predict possible drastic shifts of the states or failures, e.g. an engine blowing up, a bridge collapsin, a car consuming more gasoline etc. Solving ordinary differential equations is then an essential step of building industrial physical systems in regard to dimensioning and reliability. The solver of such ODE systems needs to be fast, accurate and relevant.

In practice, it is not possible to find a continuous function as the exact solution of the real ODE problem.
Consequently numerical methods are used to give discrete solutions which approximates the continuous one with a controllable error. The correct handline of this control is very important to get a relevant solution within an acceptable recovery time. Starting from existing studies of local and global errors, this thesis work goes more deeply and adjusts the time step of the integration time algorithm and solves the problem in a very efficient manner.

A new scheme is proposed is this thesis, to minimize the cost of integration. Another method to improve the execution speed is to parallelize the ODE solver by using a multicore and a multiprocessor architecture. Finally, the solver has been tested with different applications from OpenModelica.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France