ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Travail pratique de Master

Septembre 2004 - Janvier 2005

Résolution numérique d'équations différentielles chaotiques par un algorithme adaptatif

Lionel Walter

Chaire de Modélisation et Calcul Scientifique
Section de Mathématiques

Responsables :

Prof. Alfio Quarteroni

Dr. Erik Burman

Dr. Paolo Zunino

Table des matières

Chapitre 1

Introduction

Dans ce chapitre, nous tenterons de donner d'abord une vision intuitive des résultats que présente ce rapport. Cette vision, sans mathématiques, a ses limites, mais elle illustre assez bien tous les problèmes que nous allons étudier. Dans une deuxième partie, nous décrirons les étapes de notre travail et ses objectifs.

1.1 Métaphore météorologique

Cette section est une illustration de notre travail, mais il est clair que nos résultats peuvent être appliqués à une multitude de domaines qui n'ont rien à voir avec la météorologie. Dans la marge, nous indiquons à quel chapitre, cette comparaison intuitive se réfère.

Supposons que nous sommes lundi 8h à Lausanne et que la température extérieure est de 10 degrés. Nous voulons prévoir la température qu'il fera vendredi à 8h. Nous avons à disposition toutes les lois de la météorologie qui décrivent l'évolution de la température. Néanmoins, ces lois sont tellement complexes qu'elles sont impossibles à résoudre exactement. C'est pourquoi, nous sommes obligés de faire des approximations, en employant des méthodes numériques sur des ordinateurs. Supposons que nous ayons à disposition 10 méthodes numériques numérotées de 1 à 10. La méthode 1 est la méthode qui fournit les résultats les moins précis et la méthode 10 celle qui fournit les résultats les plus précis. Néanmoins, comme on n'a rien sans rien, la méthode 10 demande un temps de calcul beaucoup plus long que la méthode 1. Notre objectif est d'estimer la température de vendredi, à 1 degré près et en un temps de calcul minimal. Quelle méthode employer ? On a le droit d'employer des méthodes différentes pour chaque jour. Deux stratégies s'offrent à nous : Ch. 2

Stratégie A On emploie la méthode 1 pour estimer la température de mardi. Puis on emploie la méthode 2. Si la différence entre les deux températures obtenues est grande, alors on emploie la méthode 3 et ainsi de suite, jusqu'à ce que la différence entre deux températures successives soit petite. Si tel est cas, on accepte la dernière température obtenue comme température du mardi à 8h. On recommence alors le même processus pour estimer la température de mercredi à partir de la température de mardi. Nous continuons de la même façon jusqu'à vendredi. On aura donc 4 méthodes différentes, une pour estimer la température de mardi, une pour mercredi, une pour jeudi et une pour vendredi. Ch. 3

Stratégie B On emploie la méthode 1 pour estimer la température de vendredi à partir de celle de lundi. Puis l'on suppose que la température initiale n'est plus 10 degrés, mais 10.1 degrés. On réemploie la méthode 1 pour estimer la température de vendredi. Si la différence entre les

deux températures est grande, alors on fait de même avec les méthodes 2,3,... jusqu'à ce que la différence soit petite. Quand tel est le cas, on emploie la dernière méthode utilisée pour estimer la température de mardi. Supposons que le résultat soit 12 degrés. On refait alors le même raisonnement. On prend la méthode 1 et on estime la température de vendredi à partir des 12 degrés du mardi. Puis, on fait de même à partir d'une température de 12.1 degrés. On compare les résultats et on augmente le numéro de la méthode jusqu'à ce que la différence soit petite. On refait les mêmes raisonnements pour mercredi et jeudi. A nouveau, on aura donc 4 méthodes différentes, une par jour.

Mathématiquement, la stratégie A consiste à borner l'erreur locale alors que la stratégie B considère les facteurs de stabilité. La majorité des algorithmes emploient la stratégie A (c'est le cas par exemple des algorithmes de type Runge-Kutta adaptatif). Ils supposent que si l'erreur pour chaque jour est petite, alors l'erreur finale est petite. C'est souvent vrai. Néanmoins, pour des problèmes chaotiques comme la météorologie, ce n'est pas vrai. Voici 2 phénomènes présents dans les problèmes chaotiques :

L'anticyclone L'anticyclone a la particularité qu'il force la température du jour suivant à être de 15 degrés quelle que soit la température des jours précédents.

L'orage L'orage amène tout un lot de perturbations et peut beaucoup modifier la température du jour suivant. L'orage est très sensible à sa température initiale. Par exemple, il est possible qu'un orage qui arrive alors qu'il fait 10 degrés amène une température de 20 degrés le jour suivant alors qu'un orage qui arrive alors qu'il fait 9.9 degrés amène une température de 5 degrés. Une petite différence de 0.1 degré au début de l'orage a amené une grande différence de 15 degrés à la fin.

Si un anticyclone est attendu pour mercredi, il est inutile d'employer des méthodes numériques précises jusqu'à jeudi car de toute façon la température du jeudi matin sera de 15 degrés. La stratégie A ne considère pas ce fait. En effet, comme elle ne se préoccupe que de la température du lendemain, elle ne remarque pas que, par exemple, la température du mardi est très peu importante pour la température du vendredi. La stratégie B tient compte de ce type de problème.

Si un orage est attendu pour mercredi, il est important de calculer très précisément la température de mercredi. Ceci n'est pas considéré par la stratégie A. En effet, elle se contente de vérifier que la différence entre deux méthodes de numéros successifs est petite sans se soucier des grandes conséquences qu'une petite différence pourrait avoir sur la suite. La stratégie B tient compte de ce type de problème.

Néanmoins, la stratégie B a deux désavantages :

Inconvénient 1 La stratégie B demande beaucoup plus de calculs que la stratégie A. En effet elle estime toujours les températures pour le vendredi. Elle estime l'évolution lundi-vendredi, puis mardi-vendredi, mercredi-vendredi et jeudi-vendredi. 10 jours sont calculés en tout. Par contre, la stratégie A estime les évolutions lundi-mardi, mardi-mercredi, mercredi-jeudi et jeudi-vendredi. Seulement 4 jours sont calculés. Ce problème est résolu par le théorème central 3.1.1 qu'on peut illustrer comme suit.

> ***Théorème central.*** Appliquer la stratégie B à Lausanne est équivalent à appliquer la stratégie A au lieu correspondant à Lausanne sur une autre planète appelée planète duale. Sur la planète duale, le temps va du futur au passé. Le but est donc de prévoir la température du lundi sachant celle du vendredi. Les conditions météorologiques de la planète duale sont différentes de celles de la Terre. Pour passer d'une à l'autre, il faut employer une transformation appelée "Le Jacobien".

Inconvénient 2 Il se peut que les résultats de la méthode 1 avec 10 et 10.1 degrés pour le lundi donnent des résultats très proches pour le vendredi : 20 et 20.5 degrés, par exemple. Néanmoins, ces résultats sont peut-être totalement aberrants. La vraie température est peut-être de 5 degrés. La stratégie B ne s'en rend pas compte. Elle vérifie juste qu'une petite modification dans la température initiale donne une petite modification dans la température finale. Pour pallier cette déficience nous allons employer une nouvelle stratégie, la stratégie C qui est une combinaison des stratégies A et B.

Conclusion En employant la stratégie C et en utilisant le théorème central, nous pouvons estimer la température précisément, en tenant compte des anticyclones et des orages en un temps qui n'est pas plus du double de celui de la stratégie A (qui donne elle parfois de très mauvais résultats).

Pour tester la stratégie C, nous essaierons d'abord d'estimer la température dans des endroits sans anticyclones ni orages. Nous comparerons nos résultats aux vraies températures observées. Puis, nous appliquerons la stratégie C à des endroits où tous les phénomènes sont présents. Nous comparerons les résultats avec la stratégie A. Ch. 4

Puis nous essaierons d'améliorer la stratégie C. Par exemple, au lieu de tester les méthodes dans l'ordre 1,2,3,... nous déciderons par exemple de passer de la 1 à la 5 si on remarque de très fortes différences de températures. Nous remarquerons aussi que si l'on veut que la température prévue pour vendredi à Lausanne soit précise à 1 degré près, alors il est suffisant de prévoir la température sur la planète duale à 2 degrés près. Avec ces modifications, on remarquera que la stratégie C est maintenant nettement plus efficace que la stratégie A. On arrive à prédire la température à 1 degré près en un temps record. Ch. 5

Nous attaquerons alors, les prédictions de la température pour tout un pays. Ceci est nettement plus difficile car, par exemple la température de Lausanne influence celle de Genève et vice-versa. Les orages et anticyclones passent d'une à l'autre. Nous devons aussi estimer les températures dans tout un pays sur la planète duale. Nous adopterons la même démarche qu'auparavant. Nous testerons d'abord la stratégie C pour des pays où la météorologie est facile, des pays sans anticyclones ni orages. Au passage, nous en profiterons pour étudier quelques propriétés de certaines météorologies. Ch. 6

Puis nous nous attaquerons à des pays à météorologies complexes. La stratégie C se révélera fiable. Ch. 7

1.2 Aperçu du travail et objectifs

Notre objectif est de résoudre numériquement des équations différentielles ordinaires qui ont un comportement chaotique, c'est-à-dire très irrégulier et inattendu. Les méthodes les plus utilisées pour la résolution d'équations différentielles ordinaires, comme les méthodes de type "Runge-Kutta adaptatif" sont peu adaptées à de tels problèmes. Les algorithmes avec estimation a posteriori de l'erreur permettent de résoudre ces déficiences. Nous implémenterons un algorithme de ce type, appelé `mstz`, basé sur les résultats de Szepessy et al. dans [MSTZ01]. Nous traitons le problème de Cauchy suivant :

$$\begin{cases} X'(t) = a(t, X(t)) & \forall t \in [0\ T] \\ X(0) = X_0 \end{cases} \tag{1.1}$$

Nous estimons la vraie solution $X(t)$ par une approximation $\bar{X}(t)$ et nous voulons minimiser l'erreur globale au temps final $g(X(T)) - g(\bar{X}(T))$ où g est une fonction réelle donnée. `mstz` est basé sur

une représentation de l'erreur globale ayant la forme suivante :

$$\text{erreur globale} = \sum_{\text{pas de temps}} \text{erreur locale} \cdot \text{poids} = \sum_{n} e(t_n) \cdot \psi(t_n) \tag{1.2}$$

Les poids sont la solution du problème dual :

$$\begin{cases} -\dfrac{d\psi(s)}{ds} &= (a')^*(s, X(s))\ \psi(s) \\ \psi(T) &= \nabla g(X(T)) \end{cases} \tag{1.3}$$

où a' désigne le Jacobien de a par rapport à la deuxième variable et $(a')^*$ est la transposée de ce Jacobien. Par la formule (1.2), l'algorithme détermine les pas de temps où il est important de faire des calculs précis (ceux où le produit $e(t_n) \cdot \psi(t_n)$ est grand). Néanmoins, la façon de faire ces calculs est indépendante de l'algorithme.

Pour reprendre la métaphore, $e(t_n)$ est ce qui est vérifié par la stratégie A et $\psi(t_n)$ est la température sur la planète duale. On remarque donc qu'on a une combinaison des stratégies A et B.

Dans le chapitre 2, nous introduirons les notions de base sur les équations différentielles ordinaires et sur le chaos. Dans le chapitre 3, nous développerons et prouverons le *théorème central* à la base de notre algorithme adaptatif `mstz`. C'est lui qui montre que les poids satisfont (1.3). Dans le chapitre 4, nous implémenterons et validerons une version simple de `mstz` basée sur le théorème central. Comme problèmes chaotiques, nous testerons la transition à la turbulence et le problème de Lorenz. Au chapitre 5, nous améliorerons l'efficacité de `mstz` par divers ajouts. Puis, nous le comparerons à un algorithme basé sur les méthodes de Runge-Kutta adaptatives. Au chapitre 6, nous emploierons `mstz` pour résoudre des équations aux dérivées partielles, comme l'équation de la chaleur et l'équation de Burgers. Cela nous permettra d'étudier quelques propriétés de ces équations. Pour terminer, et en guise de bouquet final, nous appliquerons `mstz` à une équation aux dérivées partielles chaotique, l'équation de Kuramoto-Sivashinsky. Nous discrétiserons cette équation en espace à l'aide de la méthode pseudo-spectrale basée sur la théorie de Fourier.

Chapitre 2

Equations différentielles ordinaires et chaos

Dans ce chapitre, nous introduirons les notions de base sur les équations différentielles ordinaires (en abrégé EDO) dont nous aurons besoin au cours de ce travail. Puis nous étudierons la notion de chaos.

2.1 Le problème de Cauchy

Le problème de Cauchy (aussi appelé problème aux valeurs initiales) consiste à trouver la solution d'une EDO, scalaire ou vectorielle, satisfaisant des conditions initiales. Si $I = [0, T]$ désigne un intervalle de $\mathbb{R}$, alors le problème de Cauchy consiste à trouver une fonction $X : I \to \mathbb{R}^d$ telle que

$$\begin{cases} X'(t) = a(t, X(t)) & \forall t \in I \\ X(0) = X_0 \end{cases} \tag{2.1}$$

où $a : I \times \mathbb{R}^d \to \mathbb{R}^d$ est une fonction donnée et continue par rapport aux deux variables. $X'(t)$ désigne la dérivée par rapport à t de X qui sera aussi notée parfois $\dot{X}(t)$. La fonction $X(t)$ représente souvent l'évolution d'une quantité au cours du temps. On peut imaginer que $X(t)$ représente la température extérieure, la valeur d'une action, la position d'une planète ou encore la force qu'exerce le vent sur le mât d'un voilier. On appelle aussi ce type de problème un système dynamique.

Le théorème suivant, tiré de [Qua00], garantit l'existence et l'unicité de la solution du problème (2.1), si la fonction a est assez régulière. Nous considérons le cas scalaire ($d = 1$).

Théorème 2.1.1 (Existence et unicité) *On suppose $a(t, x)$ uniformément lipschitzienne par rapport à x, ce qui signifie qu'il existe une constante $L > 0$ t.q.*

$$|a(t, x) - a(t, y)| \leq L|x - y| \quad \forall t \in I, \quad \forall x, y \in \mathbb{R} \tag{2.2}$$

Sous cette hypothèse, le problème de Cauchy (2.1) admet une unique solution dans I. Remarquer que la condition (2.2) est automatiquement vérifiée si la dérivée de a par rapport à x est continue : en effet, dans ce cas, il suffit de prendre pour L le maximum de $|\partial a(t, x)/\partial x|$ sur $I \times \mathbb{R}$.

On ne sait résoudre analytiquement qu'un nombre très réduit d'équations différentielles ordinaires. On est donc conduit à considérer des méthodes numériques. Celles-ci peuvent en effet être appliquées

à n'importe quelle EDO, sous la seule condition qu'elle admette une unique solution. Néanmoins, une méthode numérique ne fournira qu'une solution approchée de l'EDO. Mais des estimations a posteriori de l'erreur, telles celles que nous développerons dans ce travail, peuvent permettre d'assurer que la solution numérique soit arbitrairement proche de la solution exacte.

Voici quelques définitions préliminaires :

Definition 2.1.2 ($\mathcal{O}$ de Landau)
On écrit $f(x) = \mathcal{O}(g(x))$ *si* $\lim_{x\to 0} \frac{f(x)}{g(x)} = C$ *avec* C *une constante.*
On écrit $f(x) = o(g(x))$, *si* $\lim_{x\to 0} \frac{f(x)}{g(x)} = 0$.

Exemple 2.1.3 • $x^3 + 2x^4 + x^5 = \mathcal{O}(x^3)$ *car* $\lim_{x\to 0} \frac{x^3+x^4+x^5}{x^3} = 1$
• $(x+h)^2 = x^2 + 2xh + \mathcal{O}(h^2)$ *car* $\lim_{h\to 0} \frac{(x+h)^2-x^2-2xh}{h^2} = 1$
• *si* $f(x) = \mathcal{O}(x^p)$, *alors* $f(x) = o(x^{p-1})$, *car* $\lim_{x\to 0} \frac{f(x)}{x^{p-1}} = \lim_{x\to 0} \frac{f(x)}{x^p} x = \lim_{x\to 0} Cx = 0$

Soit $\bar{X}$ une approximation discrète de X telle que $\bar{X}(0) = X_0$. Discrète signifie que la valeur de $\bar{X}$ n'est donnée que pour un nombre fini de points de l'intervalle I. Soient $0 = t_0 < t_1 < \ldots < t_N = T$ ces points. Chaque longueur $t_n - t_{n-1}$ est appelée pas de temps. N est donc le nombre de pas de temps. On définit $\Delta t_n = t_n - t_{n-1}$ et $\Delta t = \max_{n=1,\ldots,N} \Delta t_n$.

Definition 2.1.4 *Supposons que pour un* Δt *donné, une méthode numérique donne l'approximation* $\bar{X}_{\Delta t}$. *On dit que la méthode est d'ordre* p *si*

$$|X(t_n) - \bar{X}_{\Delta t}(t_n)| = \mathcal{O}\big((\Delta t)^p\big) \quad \forall n = 0, 1, \ldots, N$$

Definition 2.1.5 *Une méthode numérique pour l'approximation du problème (2.1) est dite à un pas si* $\forall n \geq 0$, $\bar{X}(t_{n+1})$ *ne dépend que de* $\bar{X}(t_n)$ *et pas de* $\bar{X}(t_j)$ *pour* $j < n$.

Definition 2.1.6 (Solution locale) $\tilde{X}(t)$ *est la solution locale du problème. Voici sa définition pour les valeurs de l'intervalle* $(t_n, t_{n+1}]$.

$$\begin{cases} \tilde{X}'(t) = a(t, \tilde{X}(t)) & \forall t \in (t_n, t_{n+1}] \\ \tilde{X}(t_n) = \bar{X}(t_n) \end{cases}$$

Notez bien qu'avec la définition ci-dessus, en général $\tilde{X}(t_n) \neq \bar{X}(t_n) \quad \forall n$.

Definition 2.1.7 (Erreur locale) *On définit* $e(t_n) = \tilde{X}(t_n) - \bar{X}(t_n)$, *l'erreur locale. Voir la figure 2.1.*

Definition 2.1.8 (Méthodes explicites et implicites) *Une méthode est dite explicite si la valeur* $\bar{X}(t_{n+1})$ *peut être calculée directement à l'aide des valeurs précédentes* $\bar{X}(t_k)$ *avec* $k \leq n$. *Une méthode est dite implicite si* $\bar{X}(t_{n+1})$ *n'est définie que par une relation implicite faisant intervenir la fonction* a.

Exemple 2.1.9 (Méthodes d'Euler) *Pour la méthode d'Euler progressive, on calcule* $\bar{X}(t_{n+1})$ *de la manière suivante.*

$$\bar{X}(t_{n+1}) = \bar{X}(t_n) + \Delta t_{n+1} a(t_n, \bar{X}(t_n))$$

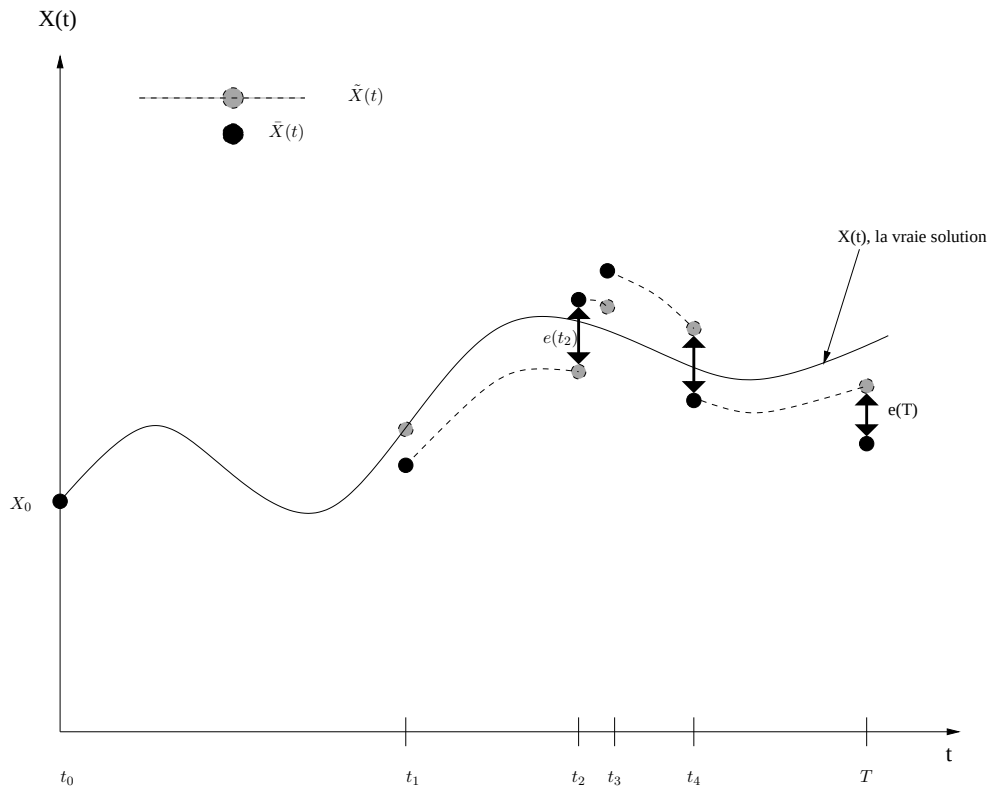


FIG. 2.1 – X, $\bar{X}$ et $\tilde{X}$

Cette méthode est donc explicite. Par contre, pour la méthode d'Euler rétrograde, on a :

$$\bar{X}(t_{n+1}) = \bar{X}(t_n) + \Delta t_{n+1} a(t_{n+1}, \bar{X}(t_{n+1}))$$

Cette méthode est donc implicite.

2.2 Notions de stabilité

Il y a deux notions de stabilité principales. Tout d'abord le problème doit être stable et ensuite la méthode numérique doit être stable. Quand on parle de stabilité d'un problème, on veut dire que si l'on perturbe un petit peu le problème, alors la solution est aussi seulement un petit peu perturbée. Si ce n'est pas le cas, il est impossible de résoudre le problème numériquement car les perturbations numériques sont inévitables. Pour une méthode numérique, le phénomène d'instabilité apparaît quand la procédure a tendance à amplifier les erreurs (comme les erreurs d'arrondis) à un tel point que ces erreurs amplifiées dominent la solution elle-même. Voici les définitions précises.

Definition 2.2.1 *On considère le problème suivant qui est une perturbation du problème (2.1).*

$$\begin{cases} Y'(t) = a(t, Y(t)) + \delta(t) & \forall t \in I \\ Y(0) = X_0 + \delta_0 \end{cases} \tag{2.3}$$

où $\delta_0 \in \mathbb{R}^d$ et $\delta : I \to \mathbb{R}^d$ est une fonction continue. Le problème de Cauchy (2.1) est stable au sens de Liapunov (ou simplement stable) sur I si, pour tout perturbation $(\delta_0, \delta(t))$ satisfaisant

$$\|\delta_0\| < \varepsilon, \qquad \|\delta(t)\| < \varepsilon \quad \forall t \in I \tag{2.4}$$

avec $\varepsilon > 0$ assez petit pour garantir l'existence de la solution du problème perturbé, alors $\exists C > 0$ indépendant de ε tel que $\|X(t) - Y(t)\| < C\varepsilon \quad \forall t \in I$.

Si a satisfait les conditions du théorème 2.1.1, alors le problème de Cauchy est stable.

Pour une méthode numérique, si la relation (2.4) est satisfaite avec une constante C indépendante de Δt, alors on dit que la méthode est stable. Plus précisément :

Definition 2.2.2 *Soit $\bar{X}_{\Delta t}$ et $\bar{Y}_{\Delta t}$ les approximations des problèmes (2.1) et (2.3) (avec $\|\delta_0\| < \varepsilon$, $\|\delta(t)\| < \varepsilon$) fournies par une méthode à un pas avec $N_{\Delta t}$ pas de temps et dont tous les pas de temps sont plus petits ou égaux à Δt. La méthode numérique est dite zéro-stable si*

$$\exists D > 0, \ \exists C > 0 \ \textit{t.q.} \ \forall \Delta t \in (0, D], \forall \varepsilon > 0, \|\bar{Y}(t_n) - \bar{X}(t_n)\| \leq C\varepsilon \quad 0 \leq n \leq N_{\Delta t}.$$

La constante C est indépendante de Δt, mais elle peut dépendre de la longueur T de l'intervalle I.

Un autre concept de stabilité est important, celui de la stabilité absolue. On dit qu'une méthode numérique est absolument stable si, pour un Δt fixé, $\bar{X}(t_n)$ reste borné quand $t_n \to \infty$. Une méthode absolument stable offre donc une garantie sur le comportement asymptotique de $\bar{X}(t_n)$, alors qu'une méthode zéro-stable assure que, pour un intervalle d'intégration fixé, $\bar{X}(t_n)$ demeure bornée quand $\Delta t \to 0$.

Considérons le problème de Cauchy linéaire suivant (scalaire, $d = 1$).

$$\begin{cases} X'(t) = \lambda X(t) & \forall t > 0 \\ X(0) = 1 \end{cases} \tag{2.5}$$

avec $\lambda \in \mathbb{C}$, dont la solution est $X(t) = \mathrm{e}^{\lambda t}$. Remarquer que si $\mathrm{Re}(\lambda) < 0$ alors $|X(t)| \to 0$ quand $t \to \infty$.

Definition 2.2.3 (Stabilité absolue) *Une méthode numérique pour l'approximation de (2.5) est absolument stable si*

$$|\bar{X}(t_n)| \to 0 \ \textit{quand} \ t_n \to \infty. \tag{2.6}$$

La région de stabilité absolue de la méthode numérique est le sous-ensemble du plan complexe

$$\mathcal{A} = \{Z = \lambda \Delta t \in \mathbb{C} \ \textit{t.q. (2.6) est vérifiée}\}$$

Souvent, les méthodes implicites ont des régions de stabilité absolue nettement plus grandes que les méthodes explicites.

2.3 Problèmes raides

Il est difficile de définir mathématiquement la raideur ("stiffness" en anglais) d'un système d'équations différentielles. C'est plutôt un phénomène, une propriété du système dont on constate les effets. Voici tout de même une définition, tirée de [Lam91].

Definition 2.3.1 *Un système d'équations différentielles ordinaires est raide s'il force une méthode numérique ayant une région de stabilité absolue de taille finie à utiliser un pas de temps excessivement petit compte tenu de la régularité de la solution exacte, quelle que soit la condition initiale pour laquelle le problème admet une solution.*

D'autres propriétés ont été observées pour les systèmes raides, mais ne sont pas vraies en toute généralité. Nous les citons tout de même pour donner une vue plus globale du phénomène de la raideur.

- Considérons un système linéaire à coefficients constants, comme $X'(t) = AX(t)$. Soit λ_i, $i = 1, \ldots, m$ les valeurs propres de A. Supposons qu'elles aient toutes une partie réelle négative et qu'elles soient numérotées t.q. $\mathrm{Re}(\lambda_1) < \mathrm{Re}(\lambda_2) < \ldots < \mathrm{Re}(\lambda_m) < 0$. Le système est raide si $\dfrac{\mathrm{Re}(\lambda_1)}{\mathrm{Re}(\lambda_m)}$ (appelé coefficient de raideur) est très grand.
- La raideur apparaît quand les contraintes de stabilité, plutôt que celles de précision, déterminent la taille des pas de temps.
- La raideur apparaît quand certaines composantes de la solution décroissent nettement plus rapidement que d'autres.

2.4 Le chaos

2.4.1 Qu'est-ce ?

Ce terme a été introduit avec sa signification actuelle en 1976, mais le début des études du chaos peut être imputé à Henri Poincaré au début du XXe siècle. C'est un concept qui apparaît dans beaucoup d'idéologies, qu'elles soient physiques, mathématiques, politiques ou religieuses. Ici, le terme chaos est employé pour indiquer que le comportement d'une solution à un problème dynamique est très irrégulier et inattendu.

Comme premier exemple, on considère une expérience réalisée par Shaw en 1984 ([Ott02]). Dans cette expérience, un lent débit d'eau est amené à un robinet. Des gouttes d'eau tombent du robinet et les intervalles de temps entre les gouttes sont enregistrés. Les données sont donc une série d'intervalles de temps Δt_1, Δt_2, ... Quand le débit d'eau est assez petit, tous les intervalles de temps Δt_n sont égaux. Quand on augmente le débit, la suite des intervalles de temps devient périodique avec un petit intervalle Δt_a suivi d'un intervalle plus long Δt_b. La suite a donc la forme ..., Δt_a, Δt_b, Δt_a, Δt_b, ... C'est une suite périodique de longueur 2. Si l'on augmente encore le débit, des suites périodiques de longueur de plus en plus grandes sont observées, jusqu'à ce que, pour un débit suffisamment grand, la séquence Δt_1, Δt_2, ... n'ait apparemment plus aucune régularité. Cette suite est dite chaotique. Ce qui est particulièrement étonnant est qu'en variant continûment un paramètre (le débit), du chaos puisse apparaître alors qu'il était complètement absent à première vue. C'est ce qu'on appelle la route vers le chaos.

2.4.2 Sensibilité aux conditions initiales

Un attribut des systèmes dynamiques chaotiques est qu'ils exhibent une sensibilité exponentielle aux conditions initiales. Considérons le problème de Cauchy (2.1) et sa solution $X(t)$ ainsi que le problème perturbé (2.3) et sa solution $Y(t)$. Néanmoins, on considère le cas où $\delta(t) = 0$, c'est-à-dire que nous ne perturbons que la condition initiale. Considérons la norme de la différence des deux solutions au temps t, $d(t) = \|Y(t) - X(t)\|$. On suppose que les solutions $X(t)$ et $Y(t)$ sont bornées

$\forall t > 0$, i.e. $\exists R > 0$ t.q $\|X(t)\| < R$ et $\|Y(t)\| < R \ \ \forall t > 0$. Si

$$\lim_{\delta_0 \to 0} \frac{d(t)}{\delta_0} \approx \mathrm{e}^{Dt} \quad D > 0$$

alors on dit que le système dépend exponentiellement des conditions initiales et est chaotique. 2 questions :

Pourquoi les solutions doivent-elles être bornées ? La raison pour cette condition est que si les solutions vont à l'infini, alors il est facile pour $d(t)$ de diverger exponentiellement. Par exemple pour l'équation scalaire $X'(t) = X(t)$ dont la solution est $X(t) = \mathrm{e}^t$ alors on a $d(t) = \delta_0 \mathrm{e}^t$ et on a divergence exponentielle et donc chaos. Mais cette solution n'est pas du tout chaotique.

Pourquoi prend-on la limite quand $\delta_0 \to 0$? Supposons que l'on demande simplement que la différence entre les solutions soit exponentielle, $d(t) \approx \mathrm{e}^{Dt}$. Vu que les solutions sont bornées, on a forcément $d(t) < 2R$ ce qui est impossible si $d(t) \approx \mathrm{e}^{Dt}$. C'est pourquoi l'on regarde le rapport $d(t)/\delta_0$ à la limite quand δ_0 tend vers 0.

La dépendance exponentielle aux conditions initiales signifie que, quand le temps avance, les petites erreurs croissent très rapidement avec le temps. Ainsi des effets comme les arrondis d'une machine peuvent totalement changer la solution par rapport à ce qu'elle serait sans ces effets. On peut voir un exemple dans la figure 2.2. Elle représente la première composante de la solution du problème de Lorenz (cf section 4.3.6) en imposant une fois la condition initiale $X(0) = 1$ et une fois $X(0) = 1.005$. On remarque qu'après un certain temps, les solutions divergent complètement.

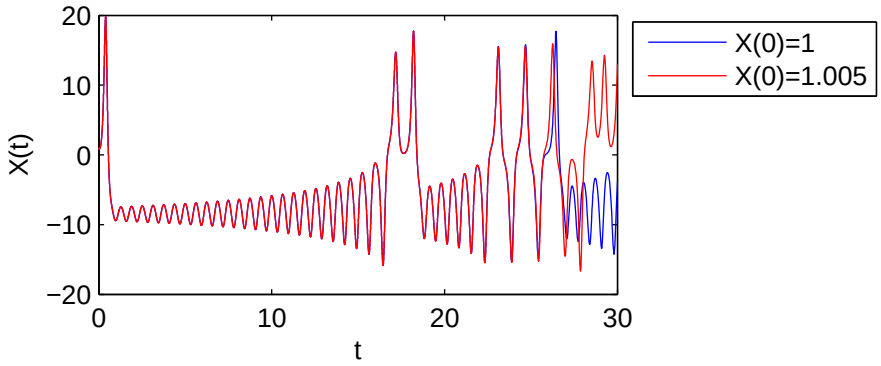


FIG. 2.2 – Sensibilité aux conditions initiales

Donc, pour un problème chaotique, après un certain temps, il devient très difficile de prédire le système. Les équations à la base des prévisions météorologiques ont un comportement chaotique, c'est pourquoi nous ne pouvons prédire le temps à long terme. Même la plus petite perturbation, comme les battements d'ailes d'un papillon, pourront finalement avoir un grand effet.

Etant donné la difficulté de faire des calculs précisément, on peut se poser la question de la validité d'illustrations comme la figure 2.2. Est-ce que la figure est juste ou est-ce plutôt l'illustration d'une erreur d'arrondi amplifiée par le chaos ? Une réponse partielle à cette question vient des preuves mathématiques de la propriété d'ombrage ("shadowing") de certains systèmes chaotiques. Par exemple, Hammel et al. ont montré dans [HYG87] que bien qu'une trajectoire numérique diverge rapidement de la vraie trajectoire avec le même point initial, il existe, sous certaines hypothèses,

une vraie trajectoire (i.e. sans erreur numérique) avec un point initial quelque peu différent qui reste proche de la trajectoire numérique pour un temps long. On a donc de bonnes raisons de croire que des figures telles 2.2 correspondent à des trajectoires réelles.

Une autre réponse partielle peut-être données par des méthodes numériques comme celle développée dans ce travail. Leur fonctionnement permet de détecter si un problème est chaotique et si c'est le cas, d'employer les techniques nécessaires à son calcul.

Remarque 2.4.1 *La notion de chaos n'est pas contradictoire avec la notion de stabilité au sens de Liapunov. Intuitivement, le chaos signifie que la constante C présente dans la définition de la stabilité au sens de Liapunov 2.2.1, grandit exponentiellement avec la longueur de l'intervalle $I = [0\ T]$.*

Chapitre 3

Un principe variationnel

Nous allons maintenant étudier le problème (2.1). Notre objectif est de minimiser l'erreur au temps final T de l'intervalle I. Par exemple, nous sommes lundi et nous voulons limiter l'erreur dans la température que nous prévoyons dans cinq jours. Nous voulons donc minimiser $X(T) - \bar{X}(T)$, ou, pour être plus général $g(X(T)) - g(\bar{X}(T))$ où g est une fonction de $\mathbb{R}^d$ dans $\mathbb{R}$. Par exemple, la température peut engendrer l'épaisseur de la couche de glace sur un lac et l'on veut minimiser l'erreur sur cette épaisseur. Ou alors $g(X(T))$ pourrait aussi désigner une composante de la solution vectorielle $X(T)$. La fonction g fait partie des données du problème.

Definition 3.0.2 *Soient $s \in [0,T]$ et $Y \in \mathbb{R}^d$. $X(t;s,Y)$ est la solution de l'équation (2.1) où l'on a remplacé la condition initiale $X(0) = X_0$ par la condition initiale $X(s) = Y$. On remarque que $X(t;0,X_0) = X(t)$.*

Definition 3.0.3

$$u : [0,T] \times \mathbb{R}^d \to \mathbb{R}$$
$$u(s,Y) = g(X(T;s,Y))$$

Autrement dit, $u(s,Y)$ mesure la valeur finale de la fonction cherchée si l'on pose la condition initiale $X(s) = Y$.

Il va nous arriver de traiter des fonctions complexes, $X : I \to \mathbb{C}^d$. Les résultats ci-dessous sont en général énoncés pour$X : I \to \mathbb{R}^d$. Néanmoins, ils se généralisent aisément au cas complexe. Lorsque ce n'est pas le cas, les résultats correspondants seront précisés.

Definition 3.0.4 *Soient U et $V \in \mathbb{R}^d$. On note*

$$\Big(U,V\Big) = \sum_{i=1}^{d} u_i v_i$$

Il en est de même dans le cas complexe. On ne prendra donc pas le conjugué du deuxième vecteur dans la somme.

Remarque 3.0.5 *Supposons que l'erreur que l'on veuille minimiser ait la forme :*

$$h(X(T)) + \int_0^T k(t, X(t))dt \tag{3.1}$$

pour des fonctions $h : \mathbb{R}^d \to \mathbb{R}$ et $k : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ données. Si l'on ajoute une équation au système (2.1) de la forme

$$y'(t) = k(t, X(t)) \text{ et } y(0) = 0$$

alors on peut définir l'objectif $g(X(T)) = h(X(T)) + y(T)$ et traiter ainsi le cas (3.1).

3.1 Le théorème central

Nous voulons trouver une méthode numérique qui permette d'assurer que l'erreur globale $g(X(T)) - g(\bar{X}(T))$ soit sous une certaine tolérance. De plus, nous aimerions arriver à cette solution le plus rapidement possible, c'est-à-dire en faisant le moins de calculs possibles. De ce point de vue, il nous serait très utile de savoir où concentrer nos forces de calculs. Quelles portions de l'intervalle $[0, T]$ sont les plus importantes en vue de minimiser l'erreur globale $g(X(T)) - g(\bar{X}(T))$.

Le théorème suivant résout exactement ce problème. En quelque sorte, c'est un théorème détecteur d'orages et d'anticyclones tels qu'on les a décrits dans le chapitre 1. L'erreur globale $g(X(T)) - g(\bar{X}(T))$ (l'erreur sur la température de vendredi) est écrite comme une somme sur les pas de temps t_n (les jours) du produit d'un poids $\psi(t_n)$ et de l'erreur locale $e(t_n)$. L'erreur locale est une fonction de notre effort de calcul. Beaucoup de calculs entre t_{n-1} et t_n vont donner une erreur locale petite alors que peu de calculs vont donner une erreur plus grande.

$$\text{erreur globale } = \sum_{\text{pas de temps}} \text{poids} \cdot \text{erreur locale} = \sum_n \psi(t_n) \cdot e(t_n)$$

Ainsi, si $\psi(t_n)$ est grand nous allons faire beaucoup de calcul pour prévoir $X(t_n)$, alors que si $\psi(t_n)$ est petit, nous n'allons faire que des calculs grossiers. Mais comment estimer les $\psi(t_n)$? Si l'on reprend l'exemple de la température développé au chapitre 1, on avait tenu le raisonnement suivant. Pour chaque jour n (lundi, mardi, mercredi, jeudi), on prend une température T_0 et on essaie d'estimer la température de vendredi à partir de la température du jour n. Puis on prend une température légèrement différente $T_0 + \delta$ et on refait le même calcul. Si les deux températures estimées pour le vendredi sont très différentes, alors on décidera de faire des calculs très précis pour le jour n. Sinon, on fera des calculs grossiers. C'est exactement ce raisonnement qui est utilisé dans le théorème suivant. Le résultat essentiel qui sera démontré est que les poids $\psi(t_n)$ satisfont une équation similaire à celle pour la température. Ils pourront donc être calculés efficacement. Ce théorème a été développé par Kyoung-Sook Moon, Anders Szepessy, Raúl Tempone et Georgios E. Zouraris dans [MSTZ01].

Cette approche donne aussi des informations sur la calculabilité ("computability") d'un problème. Est-il possible de résoudre un problème donné sur un ordinateur donné ? Dans le cas des problèmes chaotiques, cette question est très importante. Le théorème répond à cette question. En effet, supposons que notre ordinateur fasse des calculs en double précision. C'est-à-dire que la plus petite valeur que l'ordinateur peut prendre en compte est de l'ordre de 10^{-16}. Donc notre erreur locale $|e(t_n)| \geq 10^{-16}$. Si il existe un n où le poids $\psi(t_n) > 10^{16}$, disons 10^{17}, alors on a $\psi(t_n) \cdot e(t_n) \geq 10$. Il sera donc impossible d'avoir une erreur globale plus petite que 10. Autrement dit le problème n'est pas calculable, car une erreur globale plus grande que 10 est inutilisable dans la majorité des applications.

Théorème 3.1.1 (Central) *Supposons que (2.1) ait une solution unique pour toutes les valeurs initiales X_0 possibles. De plus, supposons que la fonction $a(t, x)$ soit différentiable en x, pour tout*

$t \in I$. Pour toute fonction différentiable g, l'erreur globale est une somme pondérée des erreurs locales :

$$g(X(T)) - g(\bar{X}(T)) = \sum_{n=1}^{N} \left(e(t_n), \int_0^1 \psi(t_n, \bar{X}(t_n) + se(t_n))ds \right)$$

où $\psi(s, Y) \in \mathbb{R}^d$ est la première variation de u, dans le sens que $\forall W \in \mathbb{R}^d$ et $\delta > 0$ petit :

$$u(s, Y + \delta W) - u(s, Y) = \left(\psi(s, Y), \delta W \right) + o(\delta)$$

Par exemple, si u est différentiable, on peut prendre $\psi(s, Y) = \frac{\partial}{\partial Y} u(s, Y)$.
La fonction ψ satisfait :

$$\left\{ \begin{array}{rcl} -\dfrac{d\psi(s, X(s))}{ds} & = & (a')^*(s, X(s))\ \psi(s, X(s)) \\ \psi(T, X(T)) & = & \nabla g(X(T)) \end{array} \right. \tag{3.2}$$

où $(a')^*$ est la transposée du Jacobien de a, c'est-à-dire : $[a'(s,x)]_{ij} = \frac{\partial a_i}{\partial x_j}(s,x)$, $a'(s,x) \in \mathbb{R}^{d \times d}$. $X(t)$ est la solution de (2.1). Notez bien que dans (3.2), on prend la dérivée totale par rapport à s et non la dérivée partielle par rapport à s. On appellera (3.2) le problème dual de (2.1).

Preuve 3.1.2 Par construction on a

$$u(t_n, \tilde{X}(t_n)) = u(t_{n-1}, \bar{X}(t_{n-1}))$$

En effet, $\tilde{X}(t_n)$ est le prolongement de la solution si on prend comme point de départ $\bar{X}(t_{n-1})$ (cf figure 2.1). Ceci implique :

$$\begin{array}{rcl} \displaystyle\sum_{n=1}^{N} u(t_n, \tilde{X}(t_n)) - u(t_n, \bar{X}(t_n)) & = & \displaystyle\sum_{n=1}^{N} u(t_{n-1}, \bar{X}(t_{n-1})) - u(t_n, \bar{X}(t_n)) \\ & = & u(t_0, \bar{X}(0)) - u(T, \bar{X}(T)) \\ & = & g(X(T)) - g(\bar{X}(T)) \\ & = & \text{erreur globale} \end{array}$$

Comment réécrire les termes de la somme ? On sait que $e(t_n) = \tilde{X}(t_n) - \bar{X}(t_n)$
Soit $U : [0, 1] \to \mathbb{R}$ définie par (cf figure 3.1)

$$U(s) = u(t_n, s\tilde{X}(t_n) + (1 - s)\bar{X}(t_n))$$

$$U(0) = u(t_n, \bar{X}(t_n))$$
$$U(1) = u(t_n, \tilde{X}(t_n))$$

On peut aussi écrire :

$$U(s) = u(t_n, \bar{X}(t_n) + s(\tilde{X}(t_n) - \bar{X}(t_n))) = u(t_n, \bar{X}(t_n) + se(t_n))$$

$$U(1) - U(0) = \int_0^1 U'(s)ds$$

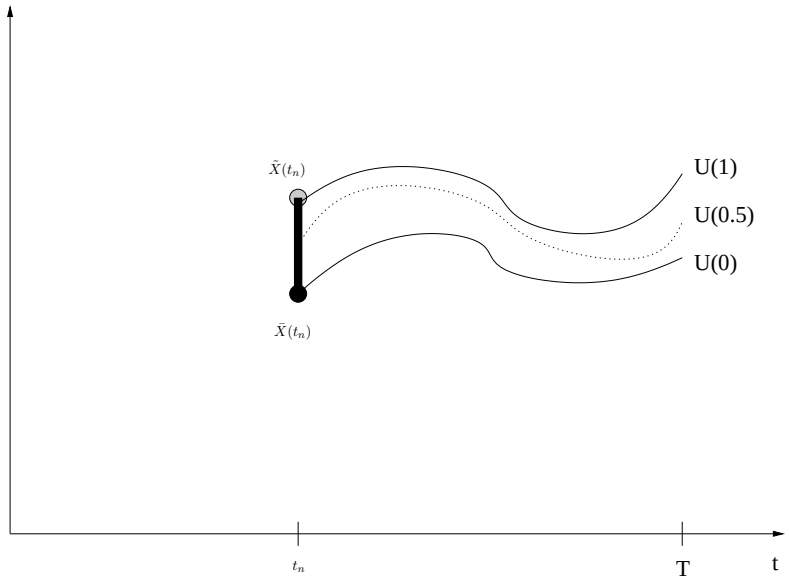


FIG. 3.1 – U, en supposant $g(x) = x$

$$U'(s) = \Big(D_2 u(t_n, \bar{X}(t_n) + se(t_n)), e(t_n) \Big)$$

où D_2 désigne la dérivée par rapport à la deuxième variable.
En combinant le tout, on obtient :

$$\begin{aligned} u(t_n, \tilde{X}(t_n)) - u(t_n, \bar{X}(t_n)) &= U(1) - U(0) = & (3.3) \\ \int_0^1 U'(s)ds &= \int_0^1 \Big(\psi(t_n, \bar{X}(t_n) + se(t_n)), e(t_n) \Big) ds & (3.4) \\ &= \Big(e(t_n), \int_0^1 \psi(t_n, \bar{X}(t_n) + se(t_n)) ds \Big) & (3.5) \end{aligned}$$

ce qui conclut la première partie de la preuve. La première variation $D_2 u(t_n, Y)$ existe car $a(t, X)$ est différentiable en X. Maintenant, nous devons vérifier que ψ satisfait bien (3.2).
Soit $X_0 \in \mathbb{R}^d$, $\delta > 0$ et $W_0 \in \mathbb{R}^d$. On définit (cf figure 3.2)

$$\begin{aligned} X_1(t) &= X(t; 0, X_0) \\ X_2(t) &= X(t; 0, X_0 + \delta W_0) \\ W(t) &= \frac{X_2(t) - X_1(t)}{\delta} \\ X_3(t; s) &= X(t; s, X_1(s)) \\ X_4(t; s) &= X(t; s, X_2(s)) \end{aligned}$$

Clairement $X_3(T; s) = X_1(T)$ et $X_4(T; s) = X_2(T)$. Ainsi la valeur finale de X_3 et X_4 ne dépend pas de s, l'endroit où l'on définit leur valeur initiale. Ainsi :

$$\frac{d}{ds} \Big[g(X_4(T; s)) - g(X_3(T; s)) \Big] = 0 \qquad (3.6)$$

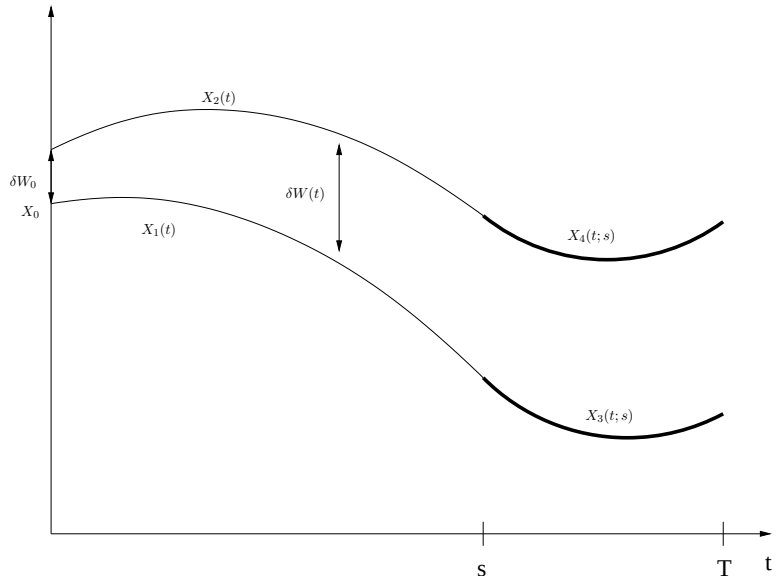


FIG. 3.2 – X_1, X_2, X_3 et X_4

Or

$$\begin{aligned} g(X_3(T;s)) &= u(s, X_1(s)) \\ g(X_4(T;s)) &= u(s, X_2(s)) \end{aligned}$$

Notons aussi que

$$\begin{aligned} \frac{d}{dt}\delta W(t) &= \frac{d}{dt}(X_2(t) - X_1(t)) && (3.7)\\ &= a(t, X_2(t)) - a(t, X_1(t)) && (3.8)\\ &= a(t, X_1(t) + \delta W(t)) - a(t, X_1(t)) && (3.9)\\ &= a(t, X_1(t)) + a'(t, X_1(t))\delta W(t) + o(\delta) - a(t, X_1(t)) && (3.10)\\ &= a'(t, X_1(t))\delta W(t) + o(\delta) && (3.11) \end{aligned}$$

De (3.6), on tire

$$\begin{aligned} 0 &= \frac{d}{ds}\big(u(s, X_2(s)) - u(s, X_1(s))\big) \\ &= \frac{d}{ds}\Big(\psi(s, X_1(s)), \delta W(s)\Big) + o(\delta) \\ &= \left(\frac{d}{ds}\psi(s, X_1(s)), \delta W(s)\right) + \left(\psi(s, X_1(s)), \frac{d}{ds}\delta W(s)\right) + o(\delta) \\ \text{(par (3.11))} &= \delta\left(\frac{d}{ds}\psi(s, X_1(s)), W(s)\right) + \left(\psi(s, X_1(s)), a'(s, X_1(s))\delta W(s)\right) + o(\delta) \end{aligned}$$

On en tire

$$\left(\frac{d}{ds}\psi(s, X_1(s)) + (a')^*(s, X_1(s))\ \psi(s, X_1(s)), W(s)\right) = \frac{o(\delta)}{\delta}$$

Ce qui implique, en prenant $\lim \delta \to 0$

$$-\frac{d}{ds}\psi(s, X_1(s)) = (a')^*(s, X_1(s))\ \psi(s, X_1(s))$$

De plus

$$\psi(s, Y) = D_2 u(s, Y) = \frac{\partial}{\partial Y} g(X(T; s, Y))$$

Donc, avec $Y = X(T)$, on a

$$\psi(T, X(T)) = \psi(T, Y) = \frac{\partial}{\partial Y} g(X(T; T, Y)) = \frac{\partial}{\partial Y} g(Y) = \nabla g(Y) = \nabla g(X(T))$$

Et ceci conclut la preuve.

□

Maintenant, nous abordons 2 exemples pour illustrer cette théorie, le premier est linéaire, c'est-à-dire de la forme $X'(t) = f(t)X(t)$, alors que le deuxième est non-linéaire.

Exemple 3.1.3

$$\begin{cases} X'(t) = X(t) & \forall t \in [0\ T] \\ X(0) = 1 \\ g(x) = x \end{cases}$$

La solution est $X(t) = e^t$. On a donc $a(t, x) = x$, $a'(t, x) = 1$ et $g'(x) = 1$. Cela donne :

$$\begin{aligned} &X(t; s, y) = y e^{t-s} \\ &u(s, y) = g(X(T; s, y)) = y e^{T-s} \\ &\psi(s, y) = \frac{\partial}{\partial y} u(s, y) = e^{T-s} \end{aligned} \tag{3.12}$$

Le système à résoudre pour ψ est :

$$\begin{cases} -\dfrac{d}{ds}\psi(s, X(s)) = \psi(s, X(s)) \\ \psi(T, X(T)) = 1 \end{cases}$$

La solution est $\psi(s, X(s)) = e^{T-s}$, ce qui correspond avec la solution vue en (3.12).

Exemple 3.1.4

$$\begin{cases} X'(t) &= 2(t+1)X^2(t) \quad \forall t \in [0\ 0.4] \\ X(0) &= 1 \\ g(x) &= x^2 \end{cases}$$

La solution est

$$X(t) = \frac{-1}{t^2 + 2t - 1}$$

On a donc : $a(t,x) = 2(t+1)x^2$, $a'(t,x) = 4(t+1)x$ *et* $g'(x) = 2x$. *Cela donne :*

$$\begin{aligned} X(t;s,y) &= \frac{-1}{t^2 - s^2 + 2t - 2s - \frac{1}{y}} \\ u(s,y) &= \left(\frac{-1}{T^2 - s^2 + 2T - 2s - \frac{1}{y}}\right)^2 \\ \psi(s,y) &= \frac{\partial}{\partial y} u(s,y) = \frac{-2}{y^2} \cdot \frac{1}{\left(T^2 - s^2 + 2T - 2s - \frac{1}{y}\right)^3} \\ \psi(s, X(s)) &= -2\frac{(s^2+2s-1)^2}{(T^2+2T-1)^3} \end{aligned} \tag{3.13}$$

Le système à résoudre pour ψ *est :*

$$\begin{cases} -\dfrac{d}{ds}\psi(s, X(s)) = \dfrac{-4(s+1)}{s^2+2s-1}\psi(s, X(s)) \\ \psi(T, X(T)) = \dfrac{-2}{T^2+2T-1} \end{cases}$$

Et on vérifie que le ψ *trouvé en (3.13) satisfait bien ce système.*

Remarque 3.1.5 *Maintenant que l'on a une représentation de l'erreur globale sous la forme d'une somme pondérée des erreurs locales, le but est d'en tirer un algorithme qui permette l'adaptation globale du maillage. Néanmoins dans l'expression*

$$\sum_{n=1}^{N} \left(e(t_n), \int_0^1 \psi(t_n, \bar{X}(t_n) + se(t_n))ds \right)$$

$e(t_n)$ *n'est pas calculable et doit être approximé par* $\bar{e}(t_n)$. *De plus,* $\psi(\cdot,\cdot)$ *est calculé comme solution du système (3.2). Or ce système ne peut être résolu analytiquement, on emploie donc une approximation* $\bar{\psi}(\cdot,\cdot)$. *On définit :*

$$\psi_n := \int_0^1 \psi(t_n, \bar{X}(t_n) + se(t_n))ds$$

et $\bar{\psi}_n$ *une approximation de* ψ_n. *On obtient donc :*

$$\sum_{n=1}^{N} \left(e(t_n), \psi_n \right) - \sum_{n=1}^{N} \left(\bar{e}(t_n), \bar{\psi}_n \right) =$$
$$\sum_{n=1}^{N} \left(e(t_n) - \bar{e}(t_n), \bar{\psi}_n \right) + \sum_{n=1}^{N} \left(e(t_n), \psi_n - \bar{\psi}_n \right)$$

Ainsi, il suffit de borner l'erreur sur $e(t_n)$ *et sur* ψ *séparément pour avoir une estimation valable. C'est ce que nous allons faire dans les deux sections suivantes.*

3.2 Approximation des poids

Pour trouver $\psi(\cdot,\cdot)$, il faut résoudre le système (3.2). Néanmoins, pour résoudre ce système, il faut avoir $X(\cdot)$ à disposition, ou du moins une approximation $\bar{X}(\cdot)$. Pour les besoins de cette section, on redéfinit $\psi(t) := \psi(t, X(t))$. On a donc :

$$-\frac{d\psi(s)}{ds} = (a')^*(s, X(s))\ \psi(s) \tag{3.14}$$
$$\psi(T) = \nabla g(X(T)) \tag{3.15}$$

Supposons que $\bar{X}$ soit une approximation de X donnée par une méthode à un pas d'ordre p. On a donc $|X(t) - \bar{X}(t)| = \mathcal{O}((\Delta t)^p)$ où Δt est comme d'habitude la taille du pas de temps maximal. Soit $\hat{\psi}$ la solution du système suivant, approximation du système (3.2) :

$$-\frac{d\hat{\psi}(s)}{ds} = (a')^*(s, \bar{X}(s))\ \hat{\psi}(s) \tag{3.16}$$
$$\hat{\psi}(T) = \nabla g(\bar{X}(T)) \tag{3.17}$$

C'est une version perturbée du problème (3.14) au même sens que le problème (2.4) était une perturbation du problème (2.1). Donc si $(a')^*$ est Lipschitz dans sa deuxième variable et que $|X(t) - \bar{X}(t)| = \mathcal{O}((\Delta t)^p)$ alors $|\psi(s) - \hat{\psi}(s)| = \mathcal{O}((\Delta t)^p)$. On résout alors le système (3.16) à l'aide de la même méthode à un pas d'ordre p utilisée pour trouver $\bar{X}$. Cela nous donne l'approximation $\bar{\psi}$ de ψ. On a donc :

$$|\bar{\psi}(s) - \psi(s)| \leq |\bar{\psi}(s) - \hat{\psi}(s)| + |\hat{\psi}(s) - \psi(s)| = \mathcal{O}((\Delta t)^p) \tag{3.18}$$

3.3 Approximation de l'erreur locale

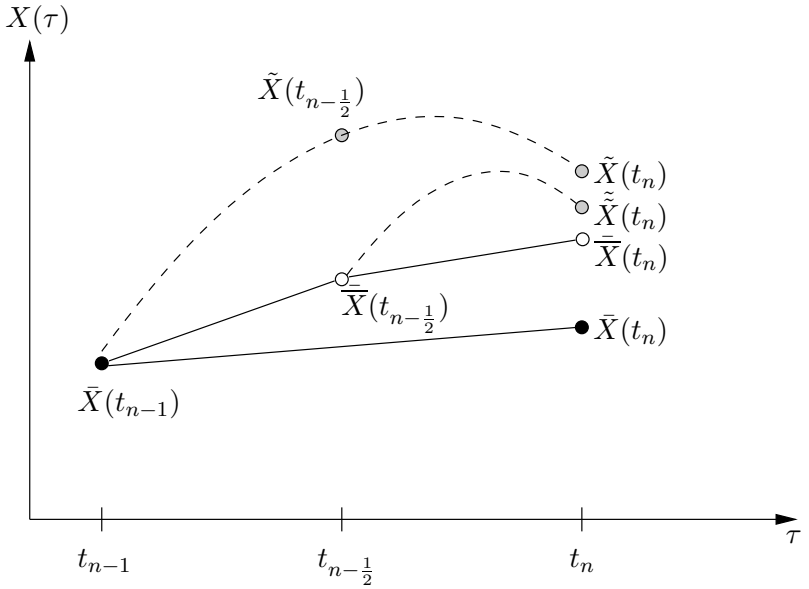


FIG. 3.3 – Les différentes variables

L'erreur locale est définie par $e(t_n) = \tilde{X}(t_n) - \bar{X}(t_n)$. Comme on ne connaît pas $\tilde{X}$, il faut l'approximer. En partant de $\bar{X}(t_{n-1})$, on emploie une approximation $\bar{\bar{X}}$ de précision différente que

$\bar{X}$, i.e. avec des pas de temps plus petits ou une méthode d'ordre plus élevé. Ensuite, on fait une extrapolation de Richardson, cf [BD72]. Par exemple, supposons que l'on utilise la même méthode d'ordre p pour $\bar{X}$ et $\overline{\bar{X}}$ mais que le pas de temps soit divisé par 2 pour $\overline{\bar{X}}$ (cf. figure 3.3). Comme la méthode est d'ordre p, l'erreur locale est d'ordre $p+1$. On a :

$$\begin{aligned}\tilde{X}(t_n) &= \bar{X}(t_n) + C(\Delta t_n)^{p+1} + \mathcal{O}((\Delta t_n)^{p+2}) \\ \tilde{X}(t_{n-\frac{1}{2}}) &= \overline{\bar{X}}(t_{n-\frac{1}{2}}) + C(\tfrac{\Delta t_n}{2})^{p+1} + \mathcal{O}((\Delta t_n)^{p+2})\end{aligned}$$

où C est une constante qui ne dépend que de $\bar{X}(t_{n-1})$. Maintenant, nous voulons trouver ce que vaut $\tilde{X}(t_n) - \overline{\bar{X}}(t_n)$. Pour cela on définit $\tilde{\tilde{X}}$ comme la solution locale, mais en partant du point $\overline{\bar{X}}(t_{n-\frac{1}{2}})$. Autrement dit :

$$\begin{cases} \tilde{\tilde{X}}'(t) = a(t, \tilde{\tilde{X}}(t)) & \forall t \in [t_{n-\frac{1}{2}}, t_n] \\ \tilde{\tilde{X}}(t_{n-\frac{1}{2}}) = \overline{\bar{X}}(t_{n-\frac{1}{2}}) \end{cases}$$

On a

$$\tilde{\tilde{X}}(t_n) = \overline{\bar{X}}(t_n) + \hat{C}(\tfrac{\Delta t_n}{2})^{p+1} + \mathcal{O}((\Delta t_n)^{p+2})$$

On a $\hat{C}$ et non C car maintenant $\hat{C}$ dépend de $\bar{X}(t_{n-\frac{1}{2}})$. Soit f, la fonction qui illustre cette dépendance. $\hat{C} = f(\bar{X}(t_{n-\frac{1}{2}}))$ et $C = f(\bar{X}(t_{n-1}))$. On peut écrire

$$\hat{C} = f(\bar{X}(t_{n-1} + \frac{\Delta t_n}{2})) = f(\bar{X}(t_{n-1})) + \mathcal{O}(\frac{\Delta t_n}{2}) = C + \mathcal{O}(\frac{\Delta t_n}{2})$$

On a donc :

$$\tilde{\tilde{X}}(t_n) = \overline{\bar{X}}(t_n) + C(\tfrac{\Delta t_n}{2})^{p+1} + \mathcal{O}((\Delta t_n)^{p+2})$$

Maintenant, il nous faut encore évaluer $\tilde{X}(t_n) - \tilde{\tilde{X}}(t_n)$. Voici un aperçu de la preuve (voir [HNW93], chapitre 10 pour plus de détails) :

$$\begin{aligned}\text{Soit } k(t) &= \tilde{X}(t) - \tilde{\tilde{X}}(t) \\ \text{On a } k(t_{n-\frac{1}{2}}) &= \tilde{X}(t_{n-\frac{1}{2}}) - \overline{\bar{X}}(t_{n-\frac{1}{2}}) \\ k'(t) &= a(t, \tilde{X}(t)) - a(t, \tilde{\tilde{X}}(t)) \\ &\leq L|\tilde{X}(t) - \tilde{\tilde{X}}(t)| \text{ en supposant } a(\cdot, x) \text{ Lipschitz} \\ &\quad \text{Si on remplace } \leq \text{ par } = \text{ et qu'on enlève la } |\cdot|, \text{ on a} \\ k'(t) &= Lk(t) \\ \Rightarrow k(t) &= k(t_{n-\frac{1}{2}})\mathrm{e}^{L(t-t_{n-\frac{1}{2}})} \\ \Rightarrow k(t_n) &= k(t_{n-\frac{1}{2}})\mathrm{e}^{L(t_n-t_{n-\frac{1}{2}})} = k(t_{n-\frac{1}{2}})(1 + \mathcal{O}(\frac{\Delta t_n}{2}))\end{aligned}$$

On obtient donc :

$$\begin{aligned}
\tilde{X}(t_n) - \bar{\overline{X}}(t_n) &= \tilde{X}(t_n) - \tilde{\bar{X}}(t_n) + \tilde{\bar{X}}(t_n) - \bar{\overline{X}}(t_n) \\
&= \big(\tilde{X}(t_{n-\frac{1}{2}}) - \tilde{\bar{X}}(t_{n-\frac{1}{2}})\big)\big(1 + \mathcal{O}(\frac{\Delta t}{2})\big) + \tilde{\bar{X}}(t_n) - \bar{\overline{X}}(t_n) \\
&= \big(C(\frac{\Delta t_n}{2})^{p+1} + \mathcal{O}((\Delta t_n)^{p+2})\big)(1 + \mathcal{O}(\frac{\Delta t}{2})) + \\
&\quad C(\frac{\Delta t_n}{2})^{p+1} + \mathcal{O}((\Delta t_n)^{p+2}) \\
&= 2C(\frac{\Delta t_n}{2})^{p+1} + \mathcal{O}((\Delta t_n)^{p+2})
\end{aligned}$$

Voici, maintenant comment nous allons estimer l'erreur locale :

$$\begin{aligned}
&\tilde{X}(t_n) = \bar{X}(t_n) + C(\Delta t_n)^{p+1} + \mathcal{O}((\Delta t_n)^{p+2}) \\
&\tilde{X}(t_n) = \bar{\overline{X}}(t_n) + 2C(\frac{\Delta t_n}{2})^{p+1} + \mathcal{O}((\Delta t_n)^{p+2}) \\
&\bar{\overline{X}}(t_n) + 2C(\frac{\Delta t_n}{2})^{p+1} = \bar{X}(t_n) + C(\Delta t_n)^{p+1} + \mathcal{O}((\Delta t_n)^{p+2}) \\
&\bar{\overline{X}}(t_n) - \bar{X}(t_n) = C(\Delta t_n)^{p+1}(1 - \frac{1}{2^p}) + \mathcal{O}((\Delta t_n)^{p+2}) \\
&e(t_n) = \tilde{X}(t_n) - \bar{X}(t_n) = C(\Delta t_n)^{p+1} + \mathcal{O}((\Delta t_n)^{p+2}) = \frac{\bar{\overline{X}}(t_n) - \bar{X}(t_n)}{1 - \frac{1}{2^p}} + \mathcal{O}((\Delta t_n)^{p+2})
\end{aligned}$$

On a donc l'approximation $\bar{e}(t_n)$ de $e(t_n)$ suivante :

$$\bar{e}(t_n) = \frac{\bar{\overline{X}}(t_n) - \bar{X}(t_n)}{1 - \frac{1}{2^p}}$$

On peut faire le même genre d'approximation si pour $\bar{\overline{X}}$ on emploie une méthode d'ordre q et pour $\bar{X}$ une méthode d'ordre p. En général, on a :

$$e(t_n) \approx \gamma(\bar{\overline{X}}(t_n) - \bar{X}(t_n))$$

où γ est appelée constante de Richardson.

Chapitre 4

L'algorithme et sa validation

Au chapitre 3, nous avons vu une expression pour le calcul de l'erreur entre une approximation $\bar{X}$ et la solution exacte X. Elle était :

$$g(X(T)) - g(\bar{X}(T)) = \sum_{n=1}^{N} \left(\bar{e}(t_n), \bar{\psi}(t_n) \right) + \mathcal{O}(\Delta t^p)$$

où $\bar{e}$ est une approximation de l'erreur locale et $\bar{\psi}$ est une approximation de la solution du problème dual. Dans ce chapitre, nous allons voir comment en tirer un algorithme de résolution d'une EDO. Nous appellerons cet algorithme `mstz` du nom des 4 auteurs du théorème 3.1.1 : Kyoung-Sook Moon, Anders Szepessy, Raúl Tempone et Georgios E.Zouraris (cf [MSTZ01]).

Nous définissons :

$$r_n = \left(\bar{e}(t_n), \bar{\psi}(t_n) \right)$$

Nous appellerons r_n le n-ème résidu. L'idée est de raffiner le maillage aux endroits où r_n est grand et de ne pas toucher (ou de déraffiner) le maillage aux endroits où r_n est petit. Le premier algorithme qui vient à l'esprit est l'algorithme `mstz` brut décrit ci-dessous. Il divise la longueur des pas de temps par 2 partout où la valeur de r_n est au-dessus d'un certain seuil.

Tous nos tests numériques seront effectués à l'aide du programme MATLAB®version 7.0.0. tournant sur un processeur Intel Pentium III à 1 Ghz.

4.1 Un algorithme basé sur le principe variationnel : `mstz`

Les variables sont les suivantes. On résout l'équation

$$\begin{cases} X'(t) = a(t, X(t)) & \forall t \in [0\ T] \\ X(0) = X_0 \end{cases}$$

$X : \mathbb{R} \to \mathbb{R}^d$. On veut que la valeur absolue de l'erreur globale $|g(X(T)) - g(\bar{X}(T))|$ ($g : \mathbb{R}^d \to \mathbb{R}$) soit sous une certaine tolérance TOL. N_0 désigne le nombre de pas de temps initiaux que nous voulons effectuer.

L'algorithme `mstz` dit où il est important de faire des calculs précis. Mais la manière de faire ces calculs est indépendante de l'algorithme. Nous emploierons des méthodes numériques à un pas (par

exemple Euler progressive, Runge-Kutta, ...). Nous n'emploierons pas de méthodes multi-pas car il est nettement plus complexe d'effectuer de l'adaptativité avec des méthodes multi-pas. Le pas de la méthode numérique utilisée est donné par $\bar{X}(t_i + \Delta t) = A(a(\cdot,\cdot), \Delta t, t_i, \bar{X}(t_i))$. $\bar{X}(t_i + \Delta t)$ est l'approximation de X au temps $t_i + \Delta t$ sachant qu'au temps t_i l'approximation vaut $\bar{X}(t_i)$. La méthode numérique est d'ordre p.

L'algorithme `mstz` brut renvoie $\mathcal{T}$ et $\bar{X}$. $\mathcal{T}$ est le maillage sur lequel l'algorithme a réussi à calculer l'approximation $\bar{X}$ qui a la propriété $|g(X(T)) - g(\bar{X}(T))| < \text{TOL}$.

Algorithme 4.1.1. `mstz` brut : divise les pas de temps par 2 si les résidus sont grands

$[\mathcal{T}, \bar{X}] =$`mstz`$(\text{TOL}, a(\cdot,\cdot), T, X_0, g(\cdot), N_0, A(\cdot,\cdot,\cdot,\cdot))$

init $\bar{X}(0) = X_0$, $0 = t_0 < \ldots < t_{N_0} = T$, un maillage uniforme de N_0 pas de temps sur $[0, T]$. $N = N_0$, le nombre de pas de temps du maillage. $\mathcal{T} = (t_0, \ldots, t_N)$.

Calcul de $\bar{X}$ On calcule $\bar{X} = (\bar{X}(t_0), \ldots, \bar{X}(t_N))$ avec la méthode définie par $A(\cdot,\cdot,\cdot,\cdot)$:

Loop Pour i de 1 à N

$\Delta t = t_i - t_{i-1}$

$\bar{X}(t_i) = A(a(\cdot,\cdot),\ \Delta t,\ t_{i-1},\ \bar{X}(t_{i-1}))$

Erreur locale On calcule $\bar{\bar{X}}$ avec un maillage 2 fois plus fin.

Loop Pour i de 1 à N

$\Delta t = t_i - t_{i-1}$

$z = A(a(\cdot,\cdot),\ \frac{\Delta t}{2},\ t_{i-1},\ \bar{X}(t_{i-1}))$

$\bar{\bar{X}}(t_i) = A(a(\cdot,\cdot), \frac{\Delta t}{2}, t_{i-1} + \frac{\Delta t}{2}, z)$

Constante de Richardson $\gamma = \frac{1}{1-0.5^p}$

Erreur $e = \gamma(\bar{\bar{X}} - \bar{X})$

Calcul de psi $\psi_N = \nabla g(\bar{X}_N)$

Loop Pour i de $N-1$ à 0

$\Delta t = t_i - t_{i+1}$ (négatif!)

$\psi_i = A(b_{\mathcal{T},\bar{X}}(\cdot,\cdot), \Delta t, t_{i+1}, \psi_{i+1})$ avec

$b_{\mathcal{T},\bar{X}}(t,\psi) = -(a')^*(t, \bar{X}(t)) \cdot \psi$ avec $\bar{X}(t)$ l'interpolation linéaire par rapport aux valeurs dans $\mathcal{T}$ et $\bar{X}$. $(a')^*(\cdot,\cdot)$ désigne la transposée du jacobien de a par rapport à la deuxième variable.

Résidus Pour tout i de 1 à N, $r_i = (e_i, \psi_i)$

Sortie si $|\sum_{i=1}^N r_i| <$TOL, stop. Fin de l'algorithme. Sinon, on continue.

Adaptation du maillage On raffine là où cela est nécessaire.

Loop Pour i de 1 à N

si $|r_i| > \dfrac{\text{TOL}}{N}$, on insère un nouveau pas de temps entre t_{i-1} et t_i.

Back Mise à jour de $\mathcal{T}$ avec les nouveaux pas de temps ajoutés. Mise à jour de N, le nombre de pas de temps. Retour au calcul de $\bar{X}$.

4.2 Algorithmes de comparaison

Pour tester la qualité de l'algorithme, nous le comparerons à deux autres algorithmes, un algorithme de raffinement uniforme basé sur le même principe variationnel que `mstz` et l'algorithme de Runge-Kutta adaptatif employant des paires d'ordre 4 et 5 dans l'estimation de l'erreur.

4.2.1 Raffinement uniforme

L'algorithme de raffinement uniforme, appelé `uniform` est identique à `mstz` sauf qu'au lieu de raffiner aux endroits où r_i est grand, il raffine partout. Autrement dit, à chaque itération le maillage a deux fois plus de pas de temps.

Algorithme 4.2.1. `uniform` : raffine partout si l'estimation de l'erreur globale est >TOL

$[\mathcal{T}, \bar{X}]$ =`uniform`$(\text{TOL}, a(\cdot,\cdot), T, X_0, g(\cdot), N_0, A(\cdot,\cdot,\cdot,\cdot))$

init $\bar{X}(0) = X_0$, $0 = t_0 < \ldots < t_{N_0} = T$, un maillage uniforme de N_0 pas de temps sur $[0, T]$. $N = N_0$, le nombre de pas de temps du maillage. $\mathcal{T} = (t_0, \ldots, t_N)$.

Calcul de $\bar{X}$ On calcule $\bar{X} = (\bar{X}(t_0), \ldots, \bar{X}(t_N))$ avec la méthode définie par $A(\cdot,\cdot,\cdot,\cdot)$:

Loop Pour i de 1 à N
$\Delta t = t_i - t_{i-1}$
$\bar{X}(t_i) = A(a(\cdot,\cdot),\ \Delta t,\ t_{i-1},\ \bar{X}(t_{i-1}))$

Erreur locale On calcule $\bar{\bar{X}}$ avec un maillage 2 fois plus fin.

Loop Pour i de 1 à N
$\Delta t = t_i - t_{i-1}$
$z = A(a(\cdot,\cdot),\ \frac{\Delta t}{2},\ t_{i-1},\ \bar{X}(t_{i-1}))$
$\bar{\bar{X}}(t_i) = A(a(\cdot,\cdot), \frac{\Delta t}{2}, t_{i-1} + \frac{\Delta t}{2}, z)$

Constante de Richardson $\gamma = \frac{1}{1-0.5^p}$

Erreur $e = \gamma(\bar{\bar{X}} - \bar{X})$

Calcul de psi $\psi_N = \nabla g(\bar{X}_N)$

Loop Pour i de $N-1$ à 0
$\Delta t = t_i - t_{i+1}$ (négatif!)
$\psi_i = A(b_{\mathcal{T},\bar{X}}(\cdot,\cdot), \Delta t, t_{i+1}, \psi_{i+1})$ avec $b_{\mathcal{T},\bar{X}}(\cdot,\cdot)$ comme pour `mstz`

Résidus Pour tout i de 1 à N, $r_i = (e_i, \psi_i)$

Sortie si $|\sum_{i=1}^{N} r_i|$ <TOL, stop. Fin de l'algorithme. Sinon, on continue.

Adaptation du maillage On divise le maillage par 2 partout. $N = 2N$ et retour au calcul de $\bar{X}$.

4.2.2 Runge-Kutta adaptatif

Quand on parle d'EDO, on pense très souvent aux méthodes de Runge-Kutta. La méthode de Runge-Kutta la plus simple n'est rien d'autre que la méthode d'Euler progressive. Un pas de cette méthode pour le problème (2.1) est donné comme suit, avec $\Delta t = t_n - t_{n-1}$:

$$\bar{X}(t_n) = A(a(\cdot,\cdot), \Delta t, t_{n-1}, \bar{X}(t_{n-1})) = \bar{X}(t_{n-1}) + \Delta t \cdot \underbrace{a(t_{n-1}, \bar{X}(t_{n-1}))}_{\approx \bar{X}'(t_{n-1})}$$

Cette méthode est d'ordre 1. Pour obtenir des méthodes d'ordre plus élevé, une des possibilités est d'évaluer la fonction a en des points intermédiaires entre t_{n-1} et t_n et d'utiliser toutes ces valeurs pour obtenir la valeur de $\bar{X}(t_n)$. C'est le principe des méthodes de Runge-Kutta. Le problème est que plus l'on veut un ordre élevé, plus le nombre d'évaluations de la fonction a doit être grand. C'est

pourquoi les méthodes que l'on emploie le plus sont celles d'ordre 4 qui demandent 4 évaluations de la fonction a. En effet, les méthodes d'ordre s avec $s \geq 5$ demandent toujours plus de s évaluations (cf [Qua00]).

Les schémas RK étant des méthodes à un pas, ils se prêtent bien à des techniques d'adaptation du pas de temps Δt, à condition que l'on dispose d'un estimateur efficace de l'erreur locale. L'estimateur d'erreur peut être construit de deux manières :

- en utilisant la même méthode RK, mais avec deux pas de discrétisation différents, comme nous l'avons fait à la section 3.3
- en utilisant deux méthodes RK d'ordre différent. En particulier, l'idéal est d'avoir deux méthodes d'ordre différent mais qui évaluent la fonction a aux mêmes endroits. Ainsi le nombre d'évaluations reste petit même si l'on emploie deux méthodes simultanément.

La deuxième manière est nettement plus efficace en terme de temps de calcul. Dormand et Prince (cf. [DP80]) ont développé une méthode très efficace implémentant cette façon de faire en utilisant une paire d'ordre 4 et 5 et requérant 6 évaluations de la fonction. Les endroits où la fonction a est évaluée sont calculés de manière à maximiser la convergence et la stabilité de la méthode. Ensuite, si l'estimation de l'erreur locale $|e|$ est inférieure à une tolérance fixée ε, on passe au pas de temps suivant. Sinon, l'estimation est répétée avec un pas de temps plus petit (la nouvelle taille dépend du rapport ε/e). Si l'estimation d'erreur est beaucoup plus petite que ε, alors au pas de temps suivant, on prendra un pas de temps plus grand.

La méthode de Dormand et Prince est implémentée dans MATLAB®. Son nom est `ode45`. En plus des conditions initiales, il faut lui fournir la tolérance ε voulue pour l'erreur locale. Cette méthode ne fournissant pas d'estimation de l'erreur globale, on l'adaptera pour donner l'algorithme `RK45`. A priori, un utilisateur de la méthode `ode45` ne peut pas savoir quelle tolérance ε demander pour avoir une erreur globale plus petite que la tolérance TOL. Ce qu'il va probablement faire est d'essayer une tolérance et de la diminuer petit à petit jusqu'à ce que le résultat $g(X(T))$ semble avoir convergé. Nous avons modélisé ce comportement dans l'algorithme ci-dessous. La tolérance initiale est ε=TOL$/N_0$. Puis l'on divise ε par 10 jusqu'à ce que la solution fournie soit satisfaisante. Pour être sûr que la solution soit bonne nous avons décidé de fournir la solution exacte au temps final $X(T)$ (ou une solution trouvée par une méthode de très haute précision si la solution exacte n'est pas disponible) à l'algorithme `RK45`. Ainsi, nous divisons ε par 10 jusqu'à ce que $|g(X(T)) - g(\bar{X}(T))| <$TOL.

Cette façon de faire implique que la comparaison n'est pas très équitable. En effet, `mstz` calcule un estimateur d'erreur alors que `RK45` n'en calcule pas, il se contente de faire la différence entre sa solution et la solution exacte qu'on lui a fourni. Pour un cas réel où la solution n'est pas connue, l'algorithme `RK45` n'est pas utilisable alors que `mstz` l'est. Néanmoins, vu que la méthode de Runge-Kutta adaptative basée sur les paires de Dormand et Prince est l'une des plus utilisée, il nous paraissait essentiel de la comparer avec `mstz`.

Algorithme 4.2.2. `RK45` : modif. d'un algo. RK adaptatif pour avoir une est. d'erreur globale

$[\mathcal{T}, \bar{X}] =$`RK45`$(\mathrm{TOL}, a(\cdot,\cdot), T, X_0, g(\cdot), N_0, X(T))$

init $k = \mathrm{TOL}/N_0$

Calcul de $\bar{X}$ $(\mathcal{T}, \bar{X}) =$`ode45`$(a(\cdot,\cdot), [0,T], X_0, \varepsilon = k)$

Calcul de l'erreur $e = g(X(T)) - g(\bar{X}(T))$

Sortie si $|e| <$TOL, stop. Fin de l'algorithme. Sinon, on continue.

Diminution de la tolérance $k = k/10$ et retour au calcul de $\bar{X}$

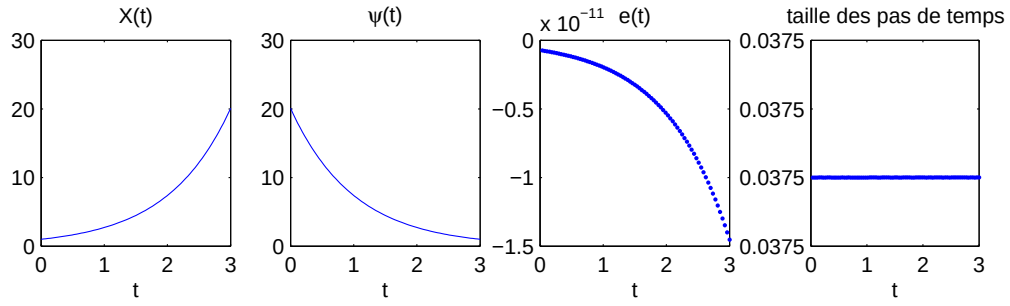
4.3 Validation

Pour tester et améliorer l'algorithme, nous nous sommes basés sur 6 problèmes tests ayant des caractéristiques bien particulières. Les figures ci-dessous sont obtenues à partir d'une application de `mstz` brut donné à la page 28 et sont là pour donner une idée au lecteur de la forme de la solution, ainsi que des endroits où l'algorithme a raffiné le maillage. Dans les tableaux, N indique le nombre de pas de temps de $\mathcal{T}$, le maillage final renvoyé par `mstz`. *it* désigne le nombre d'itérations de la méthode, i.e. le nombre de maillage différents sur lesquels on a calculé l'approximation $\bar{X}$. CPU indique en secondes le temps d'exécution de l'algorithme sur un Pentium III à 1 GHz. La méthode numérique $A(\cdot,\cdot,\cdot,\cdot)$ employée localement est une méthode de Runge-Kutta d'ordre 5. Les coefficients sont tirés de la paire de Dormand-Prince employée par l'algorithme de MATLAB®`ode45`.

4.3.1 (a) L'exponentielle

$$(a)\begin{cases} X'(t) & = X(t)\ \forall t \in [0\ 3] \\ X(0) & = 1 \\ g(x) & = x \\ \mathrm{TOL} & = 10^{-8} \\ N_0 & = 5 \end{cases} \tag{4.1}$$

C'est le problème le plus simple que l'on puisse imaginer. Il a déjà été traité dans l'exemple 3.1.3. La solution exacte est $X(t) = \mathrm{e}^t$.

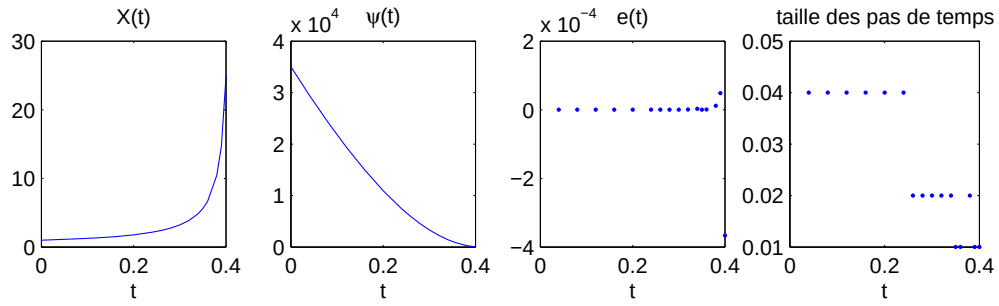


	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	1e-08	-1.16e-09	-1.16e-09	2790	930	0.85	80	5
`Uniform`	1e-08	-1.16e-09	-1.16e-09	2790	930	0.6	80	5
`RK45`	1e-08	-1.33e-09	-	776	0	0.19	79	2

4.3.2 (b) Non-linéaire avec blow-up

$$(b)\begin{cases} X'(t) & = 2(t+1)X^2(t)\ \ \forall t \in [0\ 0.4] \\ X(0) & = 1 \\ g(x) & = x^2 \\ \mathrm{TOL} & = 0.1 \\ N_0 & = 5 \end{cases} \tag{4.2}$$

Ce problème est celui de l'exemple 3.1.4. C'est un problème non-linéaire dont on connaît la solution exacte. C'est $X(t) = \frac{-1}{t^2+2t-1}$. Elle explose en $t = \sqrt{2} - 1$, ce qui permet de tester le comportement de l'algorithme proche d'une singularité de la solution. Nous remarquons que l'algorithme raffine près du point d'explosion.



	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	0.1	0.0125	0.00629	810	270	0.18	16	4
`Uniform`	0.1	-0.00224	-0.00602	1350	450	0.25	40	4
`RK45`	0.1	0.0986	-	237	0	0.06	13	3

4.3.3 (c) Stabilité

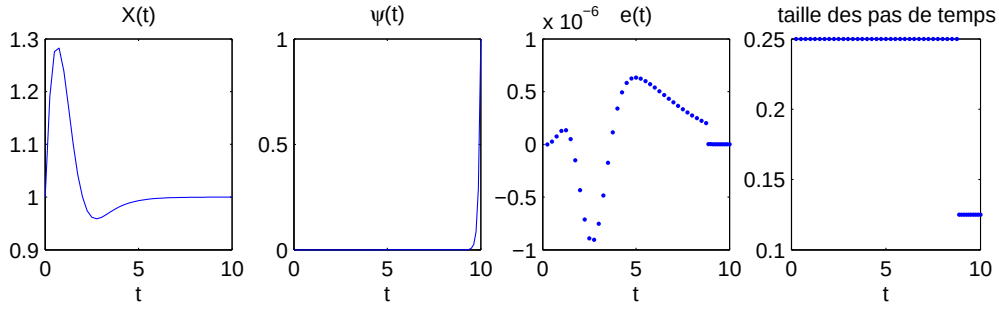
$$(c) \begin{cases} X'(t) &= t(1 - X(t)) + (1 - t)\mathrm{e}^{-t} \ \forall t \in [0\ 10] \\ X(0) &= 1 \\ g(x) &= x \\ \mathrm{TOL} &= 10^{-8} \\ N_0 &= 5 \end{cases} \tag{4.3}$$

Ce problème est tiré de [Kro73], où l'on peut trouver une liste de problèmes de référence pour tester un algorithme de résolution d'une EDO. La solution exacte de ce problème est $X(t) = \mathrm{e}^{-t^2/2} - \mathrm{e}^{-t} + 1$. Ce problème est dans la liste car il a de très mauvaises propriétés de stabilité. Comme le dit Krogh :

> *Bien que la solution devienne de plus en plus lisse quand t augmente, la taille du pas de temps doit être réduite pour maintenir une solution stable.*

C'est donc un problème raide selon la définition 2.3.1. On remarque que l'algorithme traite bien ce cas grâce au dual qui est grand pour t grand.

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	1e-08	1.39e-09	1.41e-09	2160	720	0.42	45	5
`Uniform`	1e-08	1.39e-09	1.41e-09	2790	930	0.51	80	5
`RK45`	1e-08	5.92e-10	-	895	0	0.14	146	1

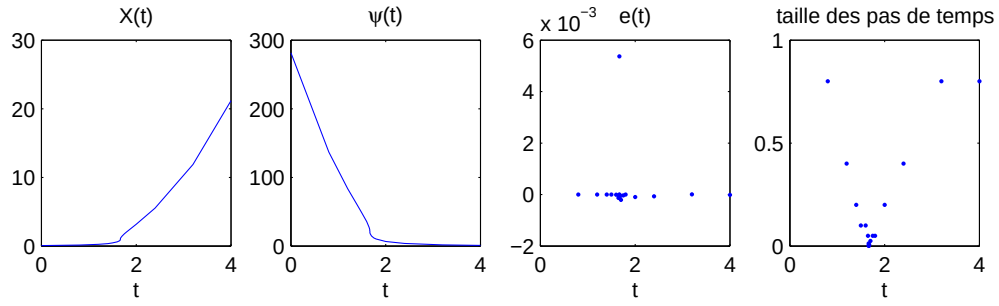


4.3.4 (d) Singularité

$$(d)\begin{cases} X'(t) &= \dfrac{X(t)}{\sqrt{t-5/3+\pi 10^{-8}}} \ \forall t \in [0\ 10] \\ X(0) &= \mathrm{e}^{-2\sqrt{\frac{5}{3}-\pi 10^{-8}}} \\ g(x) &= x \\ \mathrm{TOL} &= 0.1 \\ N_0 &= 5 \end{cases} \tag{4.4}$$

La solution exacte est $X(t) = \exp(2 \cdot \mathrm{sign}(t-5/3+\pi 10^{-8}) \cdot \sqrt{|t-5/3+\pi 10^{-8}|})$ où sign(x) vaut 1 si $x \geq 0$ et -1 sinon. Cette fonction a donc une singularité. La dérivée vaut ∞ quand $t = 5/3 + \pi 10^{-8}$. Les hypothèses du théorème central 3.1.1 ne sont donc pas vérifiées. Nous voulons voir comment l'algorithme y réagit.

D'où vient le terme étrange $\pi 10^{-8}$? Nous l'avons ajouté pour être sûr que jamais un pas de temps tombera exactement sur la singularité. Ainsi, nous n'aurons jamais une valeur valant l'infini dans l'algorithme. En effet, comme le raffinement se fait par division des pas de temps, les pas de temps ne seront toujours que des nombres rationnels.



	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	0.1	0.0484	0.0976	4320	1440	0.82	24	16
`Uniform`	-	-	-	-	-	-	-	-
`RK45`	0.1	0.0055	-	1061	0	0.2	71	5

L'algorithme `uniform` n'arrive pas à résoudre ce problème en un temps raisonnable.

4.3.5 (t) Transition à la turbulence

Voici un modèle tiré des équations de Navier-Stokes pour la dynamique des fluides. Ce modèle a été développé dans [BT97]. Il simule la transition entre un état laminaire de l'écoulement et un état turbulent. Voici le modèle, ici la dimension de X est 2 :

$$(t)\begin{cases} X'(t) &= \begin{pmatrix} -R^{-1} & 1 \\ 0 & -R^{-1} \end{pmatrix} X(t) + ||X(t)|| \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} X(t) \quad t \in [0\ 500] \\ X(0) &= \frac{\delta}{\sqrt{2}}(1,1) \quad \delta > 0 \\ g(X) &= x_1 \text{ (la première composante)} \\ \text{TOL} &= 10^{-6} \\ N_0 &= 500 \end{cases}$$

Avec $||\cdot||$, la traditionnelle 2-norme et R, le nombre de Reynolds, que l'on prendra égal à 100 dans nos tests. L'état de la solution, c'est-à-dire si elle est laminaire ou turbulente dépend de δ, c'est-à-dire de la norme de la solution initiale, comme on peut le voir dans la figure 4.2. Comme nous n'avons pas la solution exacte de ce problème, nous avons fait un calcul en demandant TOL=10^{-10}.

Pour trouver un exemple qui soit intéressant, c.à.d. dont le comportement soit assez chaotique, nous allons faire les calculs avec l'algorithme `mstz` pour les valeurs de la norme initiale entre $10^{-5.5}$ et 10^{-5}, puis nous prendrons comme exemple test, celui où le dual sera le plus élevé. Les résultats sont dans le tableau 4.1. Notre exemple de référence aura donc $10^{-5.2}$ comme valeur de δ.

Les résultats obtenus sont dans la figure 4.1.

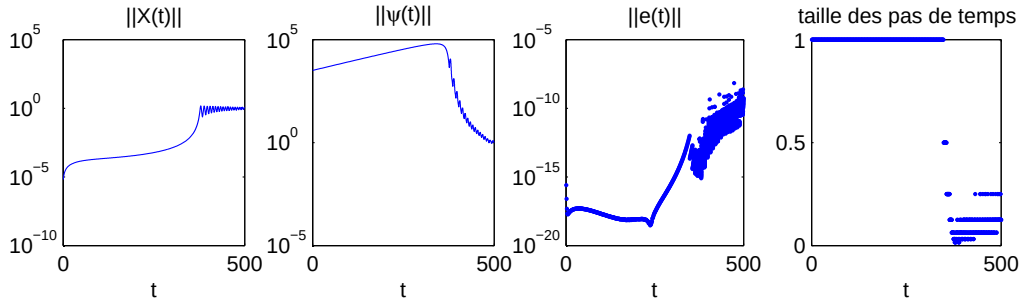


FIG. 4.1 – Transition à la turbulence. Attention, les échelles des axes des ordonnées sont logarithmiques

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	1e-06	5.49e-08	1.55e-07	184716	61572	45.6	2531	7
`Uniform`	1e-06	5.36e-09	9.51e-07	567000	189000	138.13	16000	6
`RK45`	1e-06	-9.69e-07	-	140045	0	23.98	9569	5

On peut observer l'évolution de X dans le plan $x_1 - x_2$ ainsi que du dual dans la figure 4.3. On peut remarquer que pour les temps entre 0 et 350, la solution ne bouge quasiment pas. Elle reste très proche du point (0,0). Par contre, la deuxième composante du dual grandit énormément durant cette période pour arriver proche $-8 \cdot 10^4$. Cela signifie que la 2ème composante de la solution est très importante pour la suite de la trajectoire. Puis, une fois que la solution a entamé son mouvement "circulaire", la solution n'est plus que très peu sensible aux perturbations, ce qui est signifié par un dual qui redevient très vite très petit.

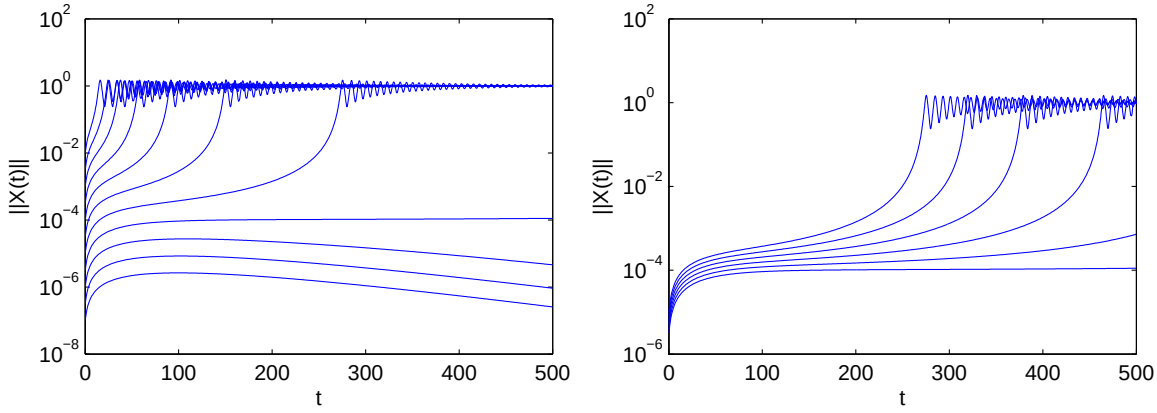


FIG. 4.2 – A gauche, l'évolution de $||X(t)||$ au cours du temps pour des valeurs de $||X_0||$ valant 10^{-7},$10^{-6.5}$, 10^{-6}, ... à 10^{-2}. A droite, la même chose pour des valeurs valant $10^{-5.5}$, $10^{-5.4}$, ..., 10^{-5}.

$\|\|X_0\|\|$	Vraie erreur	Est. erreur	$\max_{i=1,...,N} \|\|\psi_i\|\|$	# eval. a	CPU	N	it
$10^{-5.5}$	-6.52e-14	1.32e-19	1	9000	2.66	500	1
$10^{-5.4}$	1.04e-13	-4.19e-17	1	9000	2.22	500	1
$10^{-5.3}$	-4.2e-08	-7.65e-07	1576.42	60876	14.8	898	5
$10^{-5.2}$	5.49e-08	1.55e-07	62508.9	184716	45.23	2531	7
$10^{-5.1}$	-9.62e-08	-7.59e-07	40088.3	166986	41.15	2752	6
10^{-5}	-1.03e-07	-6.75e-07	34285.8	186534	46.09	3049	6

TAB. 4.1 – Test sur la transition a la turbulence

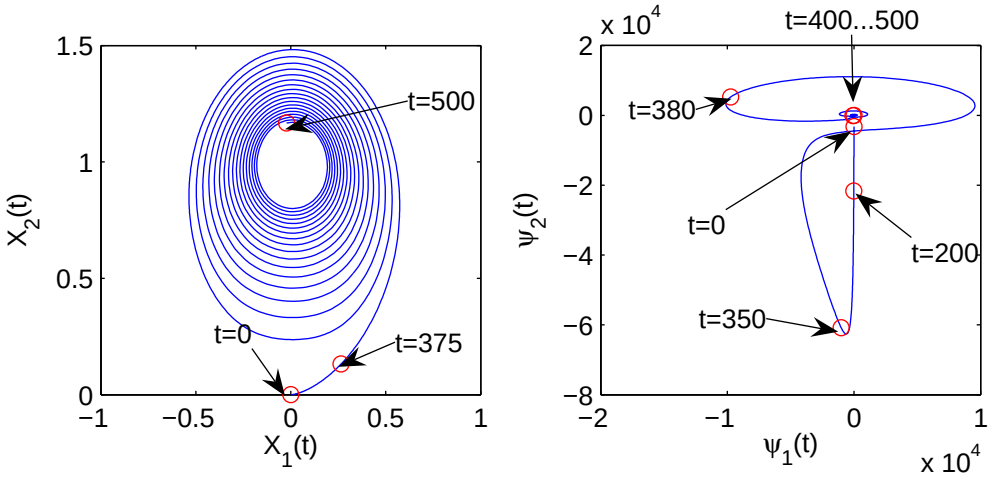


FIG. 4.3 – Les solutions du l'équation et du dual de la transition à la turbulence dans le plan $x_1 - x_2$

4.3.6 (l) Lorenz

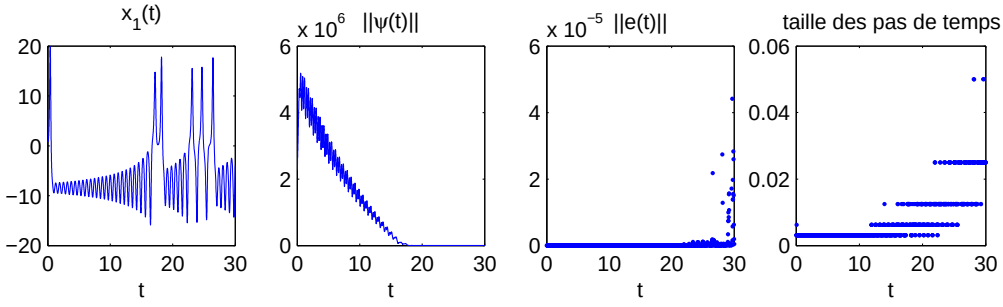
Le problème de Lorenz (qui est en 3D) a été découvert par Edward N. Lorenz à partir d'un problème météorologique. C'est une simplification d'un modèle de convection thermique. Il est connu pour son état chaotique et son extrême sensibilité aux conditions initiales. Par exemple, si l'on fait une mesure sur un intervalle de temps [0 100], alors une variation de l'ordre de 10^{-20} sur les conditions initiales donne une solution complètement différente. C'est une des raisons pour lesquelles il est encore impossible de faire des prévisions météorologiques sur plus que quelques jours. Voici le système d'equations, très simple, qui donne lieu à ce comportement extrêmement complexe. On note $X = (x_1, x_2, x_3)$.

$$(l) \begin{cases} x_1'(t) & = -\sigma x_1(t) + \sigma x_2(t) \\ x_2'(t) & = r x_1(t) - x_2(t) - x_1(t) x_3(t) \\ x_3'(t) & = x_1(t) x_2(t) - b x_3(t) \\ & \forall t \in [0\ 30] \text{ avec } \sigma = 10,\ b = 8/3,\ r = 28 \\ X(0) & = (1, 0, 0) \\ g(X) & = x_1 \\ \text{TOL} & = 0.1 \text{ et } 0.01 \\ N_0 & = 300 \end{cases} \tag{4.5}$$

En fait, ce système a en quelque sorte deux pôles. La solution oscille autour de ces pôles. Elle effectue quelques tours autour du premier, puis autour du deuxième et revient au premier et ainsi de suite. Le problème est que le nombre de tours qu'elle effectue autour de chaque pôle est très difficile à prévoir. Il suffit d'une petite erreur et l'on calcule un tour de trop ce qui donne une solution complètement différente. La figure 4.4 permet de visualiser ce phénomène.

Sur la figure 4.5, on remarque bien l'instabilité du phénomène. Sur un long intervalle de temps, il est impossible de dégager une tendance entre les oscillations autour d'un des deux pôles. Sur la figure 4.6 on peut voir la solution du dual du problème de Lorenz. Contrairement à son primal, celui-ci a un comportement très régulier : des oscillations d'amplitude de plus en plus petite autour d'un centre (pour t croissant).

Voici maitenant les résultats obtenus. La solution exacte a été calculée par une implémentation de `mstz` en FORTRAN utilisant la triple précision et une tolérance TOL de 10^{-7} (cf [MSTZ01]).



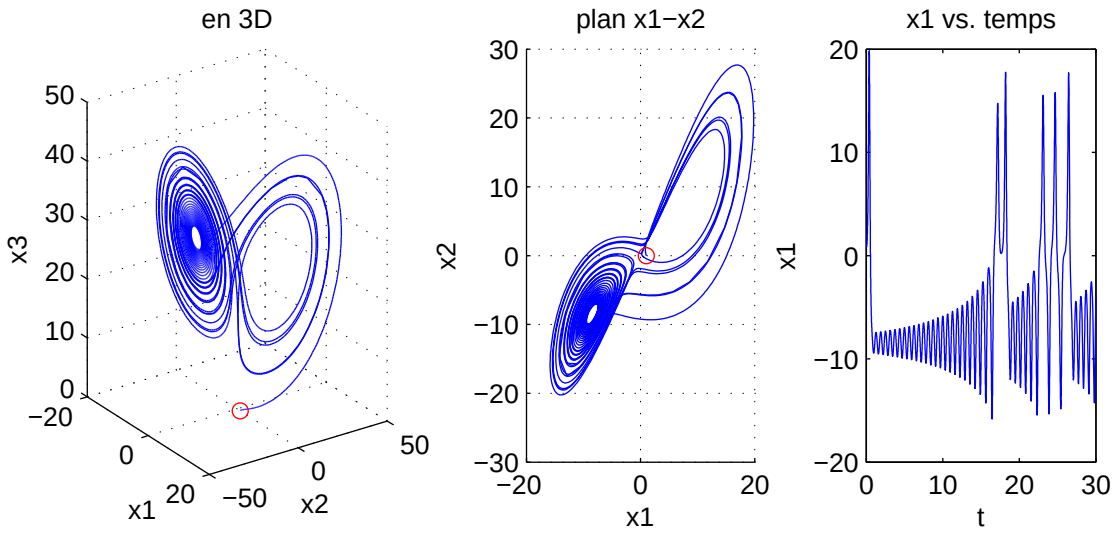


FIG. 4.4 – Solution du problème de Lorenz, le rond désigne la position initiale

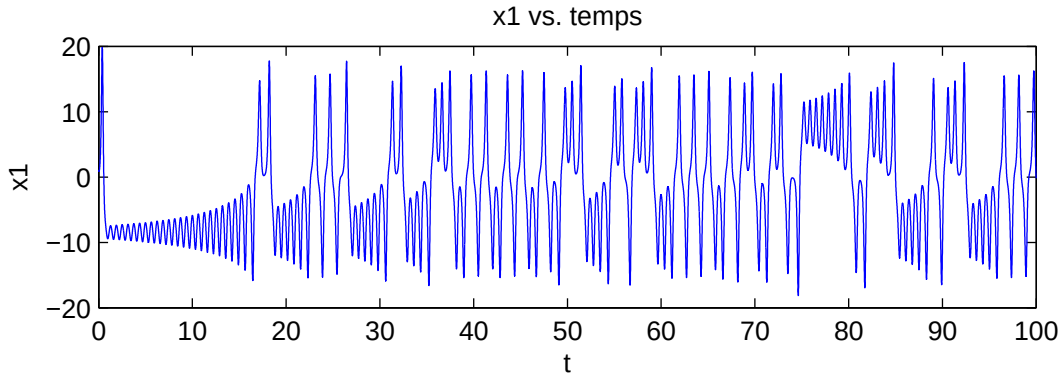


FIG. 4.5 – Allure approximative de la solution de Lorenz sur un temps plus long. On voit bien que l'on ne peut dégager aucune tendance. C'est chaotique.

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	0.1	0.0623	0.0674	256734	85578	59.27	6461	6
`Uniform`	0.1	0.0622	0.0654	340200	113400	73.98	9600	6
`RK45`	0.1	-0.0147	-	168110	0	26.11	10513	8
`mstz`	0.01	0.000979	0.000998	472788	157596	110.67	10463	7
`Uniform`	0.01	0.00191	0.00191	685800	228600	148.12	19200	7
`RK45`	0.01	-0.0015	-	264932	0	40.8	16651	8

4.4 Premiers constats

L'algorithme fonctionne. L'estimation de l'erreur est bonne. Les singularités sont gérées. L'efficacité, mesurée par le nombre d'évaluations des fonction a et $(a')^*$ est toujours meilleure pour `mstz` que pour `uniform`. Néanmoins, l'efficacité est toujours nettement plus mauvaise que celle de l'algorithme `RK45`. Par contre, le nombre de pas utilisés à la dernière itération est plus ou moins égal (exponentielle et non-linéaire avec blow-up) ou nettement plus petit (stabilité, singularité, turbulence et Lorenz). Cela montre que `mstz` raffine mieux que `RK45`. Néanmoins, l'effort qu'il doit accomplir pour trouver les bons endroits où raffiner est trop important. Il faut le réduire, par exemple en adaptant plus rapidement le maillage. Au lieu de diviser par 2 le maillage quand les r_i sont grands, il faut le diviser par un facteur plus grand, proportionnel à l'amplitude des r_i.

Un dernier gros avantage de `mstz` est qu'il fournit une estimation de l'erreur globale au temps final $g(X(T)) - g(\bar{X}(T))$ alors que `RK45` n'en fournit pas. Ceci explique aussi le coût supplémentaire de `mstz`. En effet, avec `ode45` (sur lequel est basé `RK45`), l'on fournit une tolérance pour l'erreur locale, mais l'on n'a aucune garantie sur l'erreur globale. Par exemple, il se peut que pour une erreur locale de 10^{-8}, l'algorithme n'ait pas encore convergé, i.e. que la solution fournie soit complètement différente de la vraie solution. Ceci est le cas par exemple pour le problème de Lorenz comme on peut le voir sur la figure 4.7.

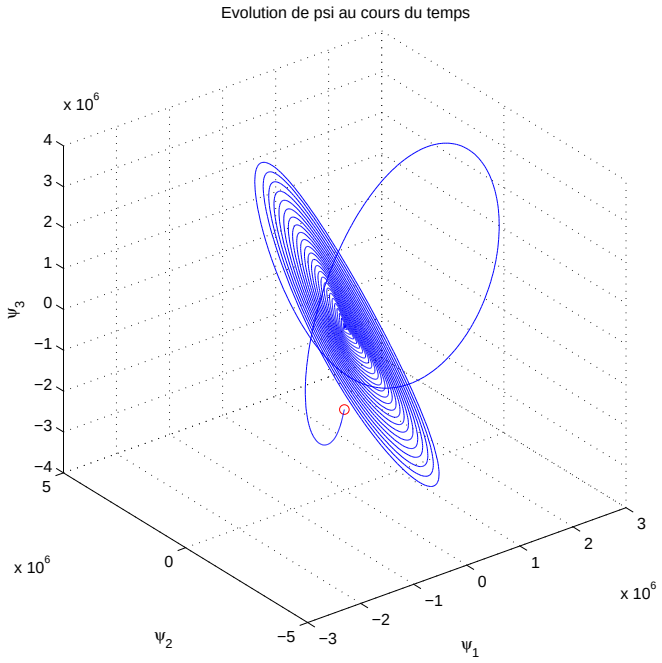


FIG. 4.6 – La solution du dual du problème de Lorenz. Vu que l'amplitude est grande au début, cela montre bien, l'importance d'une solution précise dans les premières itérations

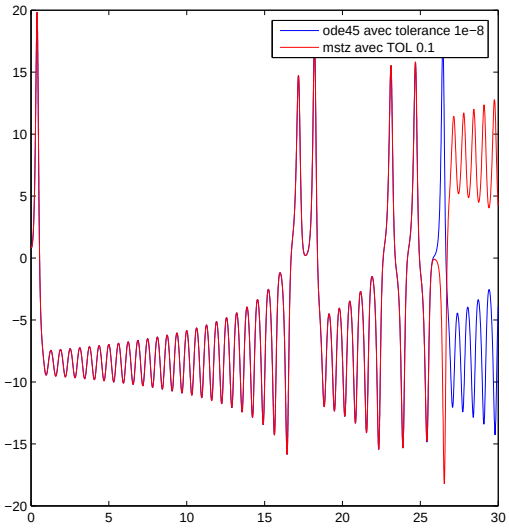


FIG. 4.7 – `ode45` est très inefficace sur le problème de Lorenz malgré une petite tolérance locale

Chapitre 5

Amélioration de l'algorithme

Nous allons maintenant essayer d'améliorer l'algorithme `mstz` brut donné à la page 28 en y ajoutant diverses fonctionnalités. Nous nous basons sur les 6 problèmes tests décrits au chapitre précédent pour améliorer l'algorithme. Les résultats finaux contenant les comparaisons des différentes versions de l'algorithme sur chaque problème test sont disponibles dans les tableaux 5.1 à 5.7, pages 49 et suivantes.

5.1 Modification du calcul du facteur de raffinement

Soit M le facteur par lequel on divise le pas de temps lorsque l'on raffine. Jusqu'à maintenant, nous avons toujours posé $M = 2$. Maintenant, nous essayons d'adapter M au cas par cas. Pour éviter d'avoir M trop grand, on le borne par une valeur M_0. De même, on prendra toujours $M \geq 2$ pour être sûr de converger. L'algorithme `mstz` brut de la page 28 sera modifié comme suit.

Algorithme 5.1.1. Modifications apportées à `mstz` brut pour adapter le facteur de raffinement

Adaptation du maillage On raffine là où cela est nécessaire.

Loop pour i de 1 à N

si $|r_i| > \dfrac{\text{TOL}}{N}$, on calcule le facteur de raffinement M selon une des manières décrites dans les prochaines sections. Puis, l'on partage l'intervalle $[t_{i-1}\ t_i]$ en $\min(\max(M, 2), M_0)$ parties égales qui formeront le maillage que l'on emploiera à l'itération suivante.

Back Mise à jour de $\mathcal{T}$ avec les nouveaux pas de temps ajoutés. Mise à jour de N, le nombre de pas de temps. Retour au calcul de $\bar{X}$.

5.1.1 Basique

La première idée est que si r_i est très grand alors on prend M grand. Comme la méthode est d'ordre p, on sait que si l'on divise le pas de temps par M, alors l'erreur locale doit être divisée grosso modo par M^{p+1}. Conclusion, si l'on veut amener r_i proche de $\frac{\text{TOL}}{N}$ il faut prendre

$$M = \left\lceil \left(\frac{|r_i|}{\frac{TOL}{N}} \right)^{\frac{1}{p+1}} \right\rceil$$

où $\lceil \cdot \rceil$ désigne la partie entière supérieure. M_0 vaut 10. Les résultats obtenus sont meilleurs, surtout pour les problèmes chaotiques : la transition à la turbulence et Lorenz.

Cette méthode a un désavantage si le meilleur raffinement est le raffinement uniforme, comme dans le cas de l'exponentielle. Si tel est le cas, alors on observe que tous les r_i sont à peu près égaux. Supposons que l'on emploie une méthode d'ordre 1, que la tolérance soit de 0.01 et qu'à une certaine étape on ait 10 pas de temps et que chaque r_i vaille 0.1. L'estimation de l'erreur est $\sum_{i=1}^{10} 0.1 = 1$. Le calcul de M donne :

$$M = \left\lceil \left(\frac{0.1}{\frac{0.01}{10}} \right)^{\frac{1}{2}} \right\rceil = 10$$

A l'étape suivante, on aura donc 100 pas de temps avec des r_i valant $\frac{0.1}{10^2} = 0.001$. L'estimation de l'erreur est $\sum_{i=1}^{100} 0.001 = 0.1$ et ce n'est pas 0.01 comme on l'aurait voulu. Comment gérer ce problème ? Si l'on divise tous les pas de temps par M et que tous les r_i sont égaux, alors l'estimation de l'erreur à l'itération suivante vaudra $NM\frac{r_i}{M^{p+1}}$. Donc le bon M pour que cette estimation vaille TOL est :

$$M = \left\lceil \left(\frac{|r_i|}{\frac{TOL}{N}} \right)^{\frac{1}{p}} \right\rceil$$

L'exposant est $\frac{1}{p}$ au lieu de $\frac{1}{p+1}$. Néanmoins, cette approche sera moins efficace lorsque c'est un pas de temps précis qui est à la base de la majorité de l'erreur comme c'est le cas dans l'exemple de la singularité (section 4.3.4). Vu que le but de notre algorithme n'est pas de résoudre des cas où le meilleur raffinement est le raffinement uniforme, nous avons décidé de garder la première version, celle avec l'exposant $\frac{1}{p+1}$. Par exemple, elle donne de meilleurs résultats sur le problème de Lorenz.

5.1.2 Calcul du maillage optimal

Une manière plus subtile de calculer le nouveau pas de temps est celle suggérée dans [MSTZ03a]. On résout un problème de calcul de variations pour trouver le pas de temps optimal, et on prend M de manière à se rapprocher le plus possible de ce pas de temps optimal.

Comme une division du pas de temps par M divise normalement r_i par M^{p+1}, la quantité $\rho_i = \frac{|r_i|}{(\Delta t_i)^{p+1}}$ devrait rester plus ou moins constante au cours des itérations, en tout cas lorsque l'on est proche de la convergence. Ce que l'on fait alors, on considère les ρ_i comme constants et on essaie de trouver le meilleur pas de temps pour atteindre la tolérance.

Pour des pas de temps donnés $0 = t_0 < \ldots < t_N = T$, on définit la fonction de maillage $\Delta t(\tau)$. Elle est constante par parties, de la manière suivante :

$$\Delta t(\tau) = \Delta t_n = t_n - t_{n-1} \quad \forall \tau \in (t_{n-1}, t_n] \text{ et } n = 1, \ldots, N$$

Ainsi le nombre de pas de temps peut être calculé comme suit :

$$N(\Delta t) = \int_0^T \frac{1}{\Delta t(\tau)} d\tau$$

En effet, si l'on regarde la fonction $\frac{1}{\Delta t(\tau)}$, l'aire de chaque rectangle au-dessus de l'intervalle $[t_{n-1}, t_n]$ est 1. Donc la somme de ces aires donne le nombre de pas de temps (cf figure 5.1).

L'estimation de la tolérance devient

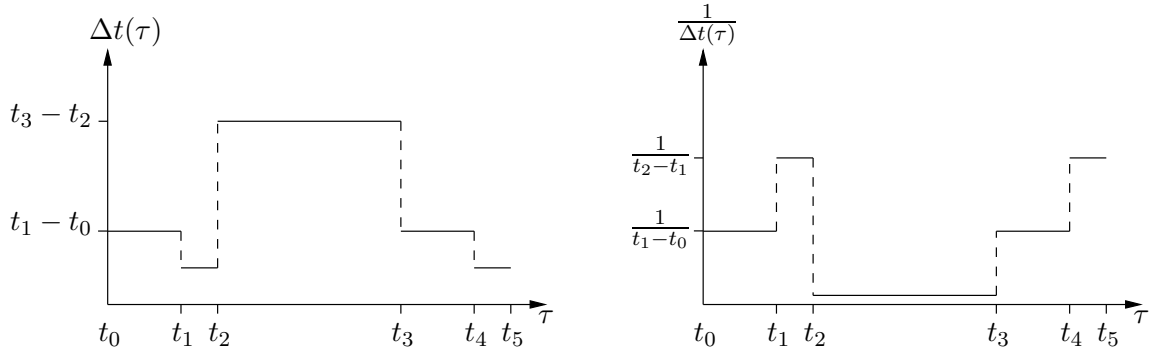


FIG. 5.1 – La fonction de maillage $\Delta t(\tau)$ et son inverse $\frac{1}{\Delta t(\tau)}$

$$\text{Erreur globale} = \sum_{i=1}^{N} r_i \leq \sum_{i=1}^{N} |r_i| = \sum_{i=1}^{N} \rho_i (\Delta t_i)^{p+1} = \int_0^T \rho(\tau) \Delta t^p(\tau) d\tau$$

Avec $\rho(\tau) = \rho_i$ si $\tau \in [t_{i-1}\ t_i]$.

Soit Δt une fonction de maillage donnée pour laquelle on a calculé les ρ_i. Le problème de minimisation à résoudre est :

$$\begin{aligned} \min_{\hat{\Delta}t \in \mathcal{K}} N(\hat{\Delta}t) &= \int_0^T \frac{1}{\hat{\Delta}t(\tau)} d\tau \\ \text{s.c.} \int_0^T \rho(\tau) \hat{\Delta}t^p(\tau) d\tau &= \text{TOL} \end{aligned}$$

avec $\mathcal{K} = \{\hat{\Delta}t \in L_2(0,T) | \hat{\Delta}t > 0$ est constante par parties sur le maillage associé à $\Delta t\}$ Autrement dit, on minimise sur des fonctions constantes par parties, positives et qui ont les mêmes noeuds que Δt.

Afin de résoudre le problème ci-dessus, on utilise les équations de Euler-Lagrange associées aux multiplicateurs de Lagrange, en suivant la démarche et les notations de [Dac92]. On définit :

$$f(\tau, u, \xi) = \frac{1}{u} - \lambda \rho(\tau) u^p$$

Si le minimum Δt^* existe, alors il satisfait

$$\begin{aligned} 0 &= f_u(\tau, \Delta t^*(\tau), (\Delta t^*)'(\tau)) \\ &= -\frac{1}{(\Delta t^*)^2(\tau)} - \lambda p \rho(\tau) (\Delta t^*)^{p-1}(\tau) \\ \Rightarrow (\Delta t^*)^{p+1}(\tau) &= -\frac{1}{\lambda p \rho(\tau)} \end{aligned}$$

Autrement dit :

$$\Delta t^*(\tau) = \frac{A}{\sqrt[p+1]{\rho(\tau)}}$$

Avec A une constante telle que $\int_0^T \rho(\tau)(\Delta t^*)^p(\tau)d\tau = \text{TOL}$. On trouve

$$A = \frac{\text{TOL}^{1/p}}{\left(\int_0^T \rho(\tau)^{\frac{1}{p+1}} d\tau\right)^{1/p}}$$

On obtient donc que la fonction de maillage optimale pour la fonction de densité ρ est

$$\Delta t^*(\tau) = \frac{1}{\rho(\tau)^{\frac{1}{p+1}}} \frac{\text{TOL}^{1/p}}{\left(\int_0^T \rho(\tau)^{\frac{1}{p+1}} d\tau\right)^{1/p}}$$

Donc, le choix de M s'effectue de la manière suivante :

$$M = \left\lceil \frac{\Delta t(\tau)}{\Delta t^*(\tau)} \right\rceil$$

Cette fois, l'on prend $M_0 = 5$, car cela donne de meilleurs résultats.

On remarque que le nombre d'évaluations total est à peu près équivalent à la méthode précédente. Souvent l'estimation de l'erreur est beaucoup plus petite que TOL. Cette différence s'explique par le fait que si pour un certain τ, on a $\Delta t^*(\tau) > \Delta t(\tau)$, on garde la valeur $\Delta t(\tau)$. Autrement dit, on n'agrandit jamais les pas de temps, on ne fait que les rétrécir. Or c'est la fonction de maillage $\Delta t^*(\tau)$ qui donne l'estimation de l'erreur TOL. Notre algorithme emploie une fonction de maillage qui s'apparente à $\min(\Delta t^*(\tau), \Delta t(\tau))$ et donc l'estimation de l'erreur est plus petite. Si l'on veut espérer avoir un algorithme efficace, il est inutile d'avoir une erreur beaucoup plus petite que TOL. Le but est d'arriver à une erreur de l'ordre de grandeur de TOL avec un nombre minimal d'évaluations de la fonction. Comment y parvenir ? 2 méthodes s'offrent à nous :

- Soit l'on raffine moins, en tenant compte du nombre de points pour lesquels Δt^* est plus grand que Δt.
- Soit l'on décide de prendre un maillage plus grossier lorsque cela est proposé par Δt^*

Nous allons expérimenter ces deux façons de faire.

Une autre raison pourrait impliquer un estimateur d'erreur si petit. L'estimateur d'erreur est $\sum r_i$. Or lorsque l'on effectue le raffinement, on se base sur $\sum |r_i| \geq \sum r_i$. Une manière de gérer ce problème serait d'effectuer le raffinement de manière différente en fonction du signe de r_i. En effet, un grand r_i négatif peut être intéressant si il est accompagné de beaucoup de petits r_i positifs. Malheureusement, nous n'avons trouvé aucune manière de gérer cela efficacement.

5.1.3 Raffinement compensé

On sait que l'on raffine seulement si $|r_i| > \frac{\text{TOL}}{N}$. Soit $\mathcal{A} = \{i \mid |r_i| > \frac{\text{TOL}}{N}\}$, c'est-à-dire l'ensemble des points où l'on raffine. Soit $\overline{\Delta t}_i$, les nouveaux intervalles de temps. Soit $\mathcal{B}$ le complémentaire de $\mathcal{A}$. On a $\overline{\Delta t}_i = \Delta t_i \ \forall i \in \mathcal{B}$, où Δt_i désigne les pas de temps à l'étape précédente. Le nouveau problème de minimisation est :

$$\begin{aligned} \min_{\overline{\Delta t} \in \mathcal{K}[k]} N(\overline{\Delta t}) &= \int_{\mathcal{A}} \frac{1}{\overline{\Delta t}(\tau)} d\tau \\ \text{s.c.} \int_{\mathcal{A}} \rho(\tau) \overline{\Delta t}^p(\tau) d\tau &= \text{TOL} - \int_{\mathcal{B}} \rho(\tau) \Delta t^p(\tau) \end{aligned}$$

La solution est la suivante :

$$\overline{\Delta t}(\tau) = \frac{1}{\rho(\tau)^{\frac{1}{p+1}}} \left(\frac{\mathrm{TOL} - \int_{\mathcal{B}} \rho(\tau) \Delta t^p(\tau)}{\int_{\mathcal{A}} |\rho(\tau)|^{\frac{1}{p+1}} d\tau} \right)^{1/p}$$

L'effet de cette technique est vraiment négligeable. On ne la gardera donc pas. Probablement que le fait d'employer $\sum |r_i|$ au lieu de $\sum r_i$ domine sur le fait d'employer $\min(\Delta t^*(\tau), \Delta t(\tau))$ plutôt que $\Delta t^*(\tau)$.

5.1.4 Déraffinement

On décide de déraffiner, c'est-à-dire de prendre un maillage plus grossier, lorsque $r_i < \frac{TOL}{N}$. Pour implémenter le déraffinement, on adopte la stratégie suivante : si $\Delta t_i^* > \Delta t_i + \Delta t_{i+1}$ et $\Delta t_{i+1}^* > \Delta t_{i+1}$, alors on assemble les intervalles $[t_{i-1}\ t_i]$ et $[t_i\ t_{i+1}]$. Sinon, l'on n'effectue aucun changement.

Le déraffinement engendre une légère amélioration des résultats, excepté pour le problème de la turbulence. Ce qu'il se passe pour ce problème est que l'algorithme déraffine pour les petits temps et raffine pour les grands. Mais finalement, il déraffine trop pour les petits temps et la méthode n'est plus stable. Ceci est détecté par un dual très grand. Alors, il raffine partout et arrive à la solution. Mais cela aura pris finalement plus d'évaluations de a que la version avec maillage optimal.

Comme les avantages de cette méthode sont minimes, nous n'approfondirons pas dans cette direction, par exemple avec un déraffinement qui assemble plus que 2 pas de temps.

5.1.5 Arrondis

Plutôt que d'arrondir vers le haut $\left(\frac{|r_i|}{\frac{\mathrm{TOL}}{N}} \right)^{\frac{1}{p+1}}$ ou $\frac{\Delta t(\tau)}{\Delta t^*(\tau)}$ pour trouver M, on arrondit vers le bas. Ceci pour la simple raison empirique que généralement `mstz` raffine trop. Cette modification donne de meilleurs résultats en termes du nombre d'évaluations de la fonction a pour tous nos exemples tests, excepté l'exponentielle. Mais dans le cas de l'exponentielle, le raffinement uniforme est le raffinement optimal et l'on sait que notre algorithme n'est pas particulièrement efficace pour ce type de problème (voir remarque à la fin de la section 5.1.1). Les résultats des tableaux 5.1 à 5.7 sont donc ceux où les arrondis sont faits vers le bas.

5.1.6 Demi-pas

Jusqu'ici la valeur de la solution calculée avec un maillage 2 fois plus fin n'était employée que pour estimer l'erreur locale. L'idée est d'employer cette estimation comme estimation de X dans l'algorithme. Autrement dit, pour calculer $\bar{X}$, on effectue deux demi-pas alors que pour estimer l'erreur locale, on emploie un pas normal. La constante de Richardson devient alors $\gamma = \frac{1}{1-2^p}$. La démarche est détaillée ci-dessous. Le raffinement est effectué de la manière basique décrite à la section 5.1.1. En effet, c'est la méthode basique qui a donné les meilleurs résultats jusqu'à maintenant.

Algorithme 5.1.2. Avec demi-pas

Calcul de $\bar{X}$ On calcule $\bar{X} = (\bar{X}(t_0), \ldots, \bar{X}(t_N))$ avec la méthode décrite par $A(\cdot,\cdot,\cdot,\cdot)$.
On emploie toutefois deux demi-pas au lieu d'un pas :

Loop Pour i de 1 à N
$\Delta t = t_i - t_{i-1}$
$z = A(a(\cdot,\cdot),\ \frac{\Delta t}{2},\ t_{i-1},\ \bar{X}(t_{i-1}))$
$\bar{X}(t_i) = A(a(\cdot,\cdot),\ \frac{\Delta t}{2},\ t_{i-1} + \frac{\Delta t}{2},\ z)$

Erreur locale On calcule $\overline{\bar{X}}$ avec un pas entier :

Loop Pour i de 1 à N
$\Delta t = t_i - t_{i-1}$
$\overline{\bar{X}}(t_i) = A(a(\cdot,\cdot), \Delta t, t_{i-1}, \bar{X}(t_{i-1}))$
Constante de Richardson $\gamma = \frac{1}{1-2^p}$
Erreur $e = \gamma(\overline{\bar{X}} - \bar{X})$

Une particularité de cette méthode est que l'on résout le dual avec une précision moins grande que le primal. En effet, on emploie des demi-pas pour le primal mais que des pas entiers pour le dual.

On obtient de très bons résultats en terme de nombre d'évaluations de la fonction a. Par contre, dans le cas de la singularité, l'estimation de l'erreur est très mauvaise. C'est un résultat logique. En effet, avant l'utilisation du demi-pas, on estimait l'erreur à l'aide d'une estimation de X de meilleure précision, puisqu'on estimait l'erreur à l'aide d'un maillage 2 fois plus fin. Maintenant, on estime l'erreur avec une estimation de X sur un maillage deux fois plus grossier. L'estimation est donc moins bonne.

Ces deux estimations sont précises à l'ordre $\mathcal{O}((\Delta t_n)^{p+2})$. On a :

$$e(t_n) = \bar{e}(t_n) + C(\Delta t_n)^{p+2} + \mathcal{O}((\Delta t_n)^{p+3})$$

où la constante C est proportionnelle au maximum de la dérivée de X par rapport à t sur l'intervalle $[t_{n-1}\ t_n]$. Or ce maximum vaut ∞ dans le cas de la singularité, c'est pourquoi l'estimation de l'erreur est si mauvaise dans ce cas.

5.2 Résultats finaux

On remarque que les améliorations par rapport à l'algorithme `mstz` brut sont notables. Sur le problème de Lorenz, le nombre d'évaluations de la fonction a a diminué de 63% pour la tolérance 0.1 et de 75% pour la tolérance 0.01. Pour la transition à la turbulence, l'amélioration est de 72%.

De plus, on arrive à faire nettement mieux que l'algorithme `RK45`. Sur le problème de Lorenz, `mstz` (demi-pas) emploie 43% d'évaluations de a de moins que `RK45` pour la tolérance 0.1 et 56% pour la tolérance 0.01. Pour la transition à la turbulence, l'amélioration est de 63%.

C'est la méthode demi-pas qui donne les meilleurs résultats. Sur le problème de Lorenz, on remarque que la version basique est meilleure que les versions avec calcul du maillage optimal sur la tolérance 0.1 alors que c'est le contraire sur la tolérance 0.01. Noter que pour la version basique la borne pour M vaut 10 alors que pour le maillage optimal elle vaut 5. Ce sont ces valeurs qui ont donné les meilleurs résultats. En fait, la version basique et celle avec calcul du maillage optimal sont liées. Soit M_b le facteur de division de la version basique et M_o le facteur de division pour la fonction avec recherche de maillage optimal. On a

$$M_b = \left\lfloor \left(\frac{|r_i|}{\frac{TOL}{N}} \right)^{\frac{1}{p+1}} \right\rfloor \qquad M_o = \left\lfloor \frac{\Delta t_i}{\overline{\Delta t_i}} \right\rfloor$$

Or :

$$\frac{|r_i|}{\frac{TOL}{N}} = \frac{N|r_i|}{TOL} = \frac{N\Delta t_i^{p+1}\rho_i}{TOL} \tag{5.1}$$

$$\Rightarrow \left(\frac{|r_i|}{\frac{TOL}{N}}\right)^{\frac{1}{p+1}} = \left(\frac{N}{TOL}\right)^{\frac{1}{p+1}} \frac{\Delta t_i}{\Delta t_i^*} \left(\frac{\mathrm{TOL}}{\int_0^T \rho(\tau)^{\frac{1}{p+1}} d\tau}\right)^{1/p} \tag{5.2}$$

le dernier membre à droite étant dépendant de l'itération. On a donc :

$$M_b = \mathrm{cste} \cdot M_o$$

La constante étant donnée par (5.2).

Dans la figure 5.2, il y a un résumé graphique de toutes les méthodes. On remarque bien la supériorité de la méthode avec demi-pas par rapport aux autres méthodes. Ensuite vient `RK45`. Puis toutes les autres méthodes sont à peu près équivalentes, à part `mstz` brut qui est bon dernier.

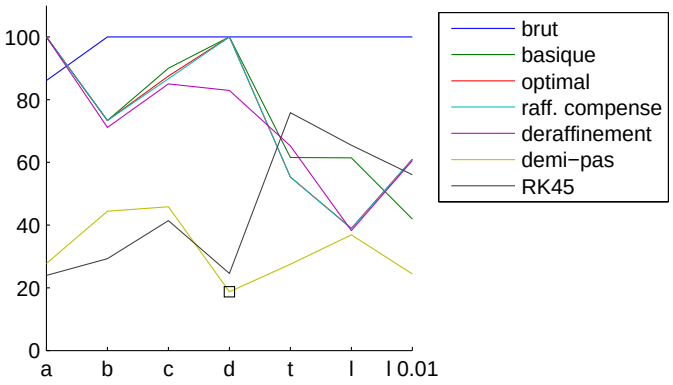


FIG. 5.2 – Comparaison des différentes méthodes sur les 6 problèmes tests. L'axe des x représente nos 6 problèmes tests. L'axe des y représente le nombre d'évaluations de la fonction a normalisé de manière à ce que le maximum soit 100. Le carré pour la méthode demi-pas, problème test "d" indique que pour ce point l'algorithme a échoué, l'erreur étant au-dessus de la tolérance. C'est le seul cas où l'algorithme échoue.

Une chose est sûre : pour que l'algorithme `mstz` soit plus efficace que `RK45`, il faut que le dual prenne de très grandes valeurs, c'est-à-dire que le problème soit chaotique. Si le dual est de l'ordre de 100, alors le problème n'est pas assez chaotique pour que l'algorithme `mstz` soit nettement plus efficace que `RK45`. Dans de tels contextes, les deux algorithmes auront à peu près le même comportement car ils vont tous deux borner l'erreur locale.

Dans la suite de ce travail, nous emploierons toujours la version avec demi-pas car c'est elle qui est la meilleure si la fonction a n'a pas de singularité.

Pour terminer, nous comparons les résultats de la méthode demi-pas avec ceux obtenus par Szepessy et al. dans [MSTZ03a]. Le problème testé est le problème de Lorenz. Szepessy et al. emploient aussi une méthode de Runge-Kutta d'ordre 5 comme méthode locale. Le nombre de pas de temps initial est identique. Par contre, ils n'ont pas de modification du facteur de raffinement. Ils divisent les pas de temps par 2, là où cela est nécessaire. De plus, ils n'emploient pas de demi-pas. Leur algorithme a quelques autres différences avec le nôtre, cf [MSTZ03a] pour les détails. Leur critère

de comparaison est le nombre de pas de temps N à l'itération finale et le nombre de pas de temps N_{tot} utilisés en tout, i.e. la somme des nombres de pas de temps employés à chaque itération. Pour faire une comparaison équitable, nous multiplierons le nombre de pas de temps employés par la méthode avec demi-pas par 2, car l'approximation $\bar{X}$ est bel et bien calculée sur $2N$ pas de temps. La comparaison est dans le tableau 5.8.

On remarque que nos résultats sont nettement meilleurs. Le nombre de pas de temps N à la dernière itération est à peu près équivalent. Néanmoins, le nombre de pas de temps total N_{tot} utilisés est 2 à 3 fois plus petit pour la méthode `mstz` avec demi-pas. Leur méthode s'apparente à l'algorithme `mstz` brut.

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
brut	1e-08	-1.16e-09	-1.16e-09	2790	930	0.85	80	5
basique	1e-08	-3.86e-10	-3.86e-10	3240	1080	0.61	100	4
avec Δt^*	1e-08	-3.86e-10	-3.86e-10	3240	1080	0.63	100	4
raff. comp.	1e-08	-3.86e-10	-3.86e-10	3240	1080	0.64	100	4
déraffinement	1e-08	-3.86e-10	-3.86e-10	3240	1080	0.66	100	4
demi-pas	1e-08	-4.8e-09	-4.38e-09	900	300	0.19	30	3
`RK45`	1e-08	-1.33e-09	-	776	0	0.19	79	2

TAB. 5.1 – L'exponentielle

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
brut	0.1	0.0125	0.00629	810	270	0.18	16	4
basique	0.1	0.0125	0.00629	594	198	0.12	16	3
avec Δt^*	0.1	0.0125	0.00629	594	198	0.13	16	3
raff. comp.	0.1	0.0125	0.00629	594	198	0.13	16	3
déraffinement	0.1	0.0176	0.012	576	192	0.11	15	3
demi-pas	0.1	0.0194	-0.0122	360	120	0.07	8	3
`RK45`	0.1	0.0986	-	237	0	0.07	13	3

TAB. 5.2 – Non-linéaire avec blow-up

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
brut	1e-08	1.39e-09	1.41e-09	2160	720	0.43	45	5
basique	1e-08	4.77e-10	4.87e-10	1944	648	0.37	53	3
avec Δt^*	1e-08	1.01e-10	1.02e-10	1890	630	0.36	75	3
raff. comp.	1e-08	1.01e-10	1.02e-10	1872	624	0.35	74	3
déraffinement	1e-08	1.01e-10	1.02e-10	1836	612	0.34	72	3
demi-pas	1e-08	3.68e-10	8.41e-10	990	330	0.19	50	2
`RK45`	1e-08	5.92e-10	-	895	0	0.13	146	1

TAB. 5.3 – Stabilité

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
brut	0.1	0.0484	0.0976	4320	1440	0.84	24	16
basique	0.1	0.0484	0.0976	4320	1440	0.9	24	16
avec Δt^*	0.1	0.0484	0.0976	4320	1440	0.89	24	16
raff. comp.	0.1	0.0484	0.0976	4320	1440	0.86	24	16
déraffinement	0.1	0.00708	0.0607	3582	1194	0.72	22	16
demi-pas	0.1	**-1.1**	**0.0858**	810	270	0.16	10	6
`RK45`	0.1	0.0055	-	1061	0	0.2	71	5

TAB. 5.4 – Singularité

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
brut	1e-06	5.49e-08	1.55e-07	184716	61572	45.37	2531	7
basique	1e-06	6.31e-08	1.7e-07	113652	37884	27.81	2514	4
avec Δt^*	1e-06	7.69e-08	2.56e-07	102150	34050	25.35	2402	4
raff. comp.	1e-06	7.56e-08	2.52e-07	102186	34062	25	2407	4
déraffinement	1e-06	8.99e-08	1.46e-07	120528	40176	30.3	3843	4
demi-pas	1e-06	9.16e-08	3.19e-07	50832	16944	12.54	1335	3
`RK45`	1e-06	-9.69e-07	-	140045	0	24.26	9569	5

TAB. 5.5 – Transition à la turbulence

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
brut	0.1	0.0623	0.0674	256734	85578	59.68	6461	6
basique	0.1	-0.0251	-0.0261	157680	52560	35.81	5789	3
avec Δt^*	0.1	0.0231	0.0282	100080	33360	22.49	3774	3
raff. comp.	0.1	0.0158	0.0197	99774	33258	22.42	3757	3
déraffinement	0.1	-0.00383	-0.000287	98190	32730	22.23	3669	3
demi-pas	0.1	-0.0146	-0.0182	94716	31572	21.44	3176	3
`RK45`	0.1	-0.0147	-	168110	0	25.83	10513	8

TAB. 5.6 – Lorenz, tolérance 0.1

	TOL	Vraie erreur	Est. erreur	# eval. a	# eval. $(a')^*$	CPU	N	it
brut	0.01	0.000979	0.000998	472788	157596	113.38	10463	7
basique	0.01	-0.0037	-0.00365	198108	66036	45.97	7949	3
avec Δt^*	0.01	-0.000761	-0.000862	288630	96210	69.87	9265	4
raff. comp.	0.01	-0.000756	-0.000856	288504	96168	69.69	9261	4
déraffinement	0.01	-0.000565	-0.000605	286416	95472	68.42	9200	4
demi-pas	0.01	-0.00428	-0.00521	115434	38478	26.68	3985	3
`RK45`	0.01	-0.0015	-	264932	0	41.99	16651	8

TAB. 5.7 – Lorenz, tolérance 0.01

	TOL	|Vraie erreur|	N	N_{tot}
`mstz` demi-pas	0.1	0.02	6352	10524
`mstz` Szepessy	0.1	0.01	6000	20000
`mstz` demi-pas	0.01	0.004	7970	12826
`mstz` Szepessy	0.01	0.003	9000	34000

TAB. 5.8 – Comparaison entre notre algorithme et l'algorithme développé par Szepessy et al. dans [MSTZ03a] sur le problème de Lorenz

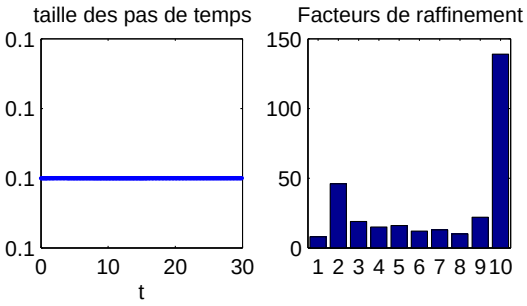
5.3 Analyse de RK45 et mstz sur le problème de Lorenz

Nous considérerons dans cette section le cas où la tolérance est 0.01. Par quelle magie l'algorithme `mstz` arrive-t-il à résoudre le problème avec seulement 115'434 évaluations de la fonction a alors que `RK45` en a besoin de 264'932 ? Pour ce faire, nous allons analyser l'évolution de ces deux algorithmes sur le problème. Nous compterons les évaluations de a et $(a')^*$ simultanément. Le nombre entre parenthèses indiquera le nombre d'évaluations totales. L'histogramme des facteurs de raffinement indique le nombre de pas de temps que l'algorithme a divisé par 1,2,...,10 à chaque itération.

5.3.1 mstz avec demi-pas

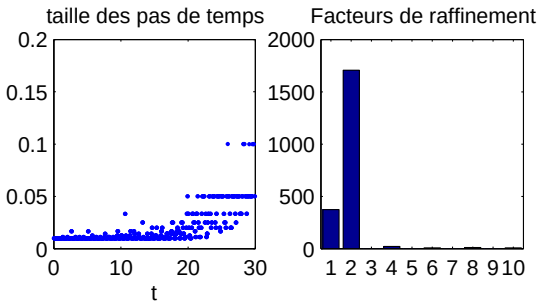
itération #1

nombre de pas de temps :	300
# évaluations pour calculer $\bar{X}$:	3600 (3600)
# évaluations pour calculer $\bar{e}$:	1800 (5400)
# évaluations pour calculer $\bar{\psi}$:	1800 (7200)
estimation de l'erreur globale :	7.56



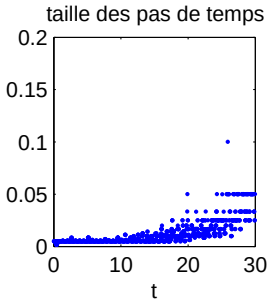
itération #2

nombre de pas de temps :	2128
# évaluations pour calculer $\bar{X}$:	25'536 (32'736)
# évaluations pour calculer $\bar{e}$:	12'768 (45'504)
# évaluations pour calculer $\bar{\psi}$:	12'768 (58'272)
estimation de l'erreur globale :	0.87



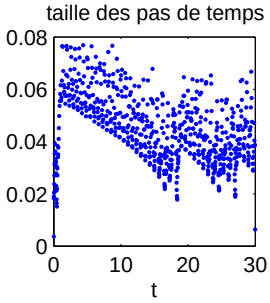
itération #3

nombre de pas de temps :	3985
# évaluations pour calculer $\bar{X}$:	47'820 (106'092)
# évaluations pour calculer $\bar{e}$:	23'910 (130'002)
# évaluations pour calculer $\bar{\psi}$:	23'910 (153'912)
estimation de l'erreur globale :	-0.0042

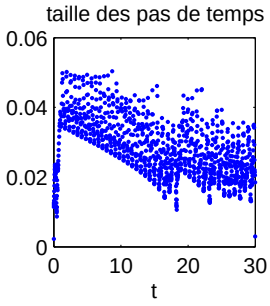


5.3.2 RK45

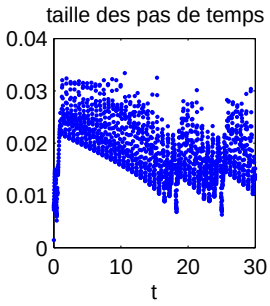
itération #1
nombre de pas de temps : 678
évaluations pour calculer $\bar{X}$: 4255(4255)
borne pour l'erreur locale ε : $1/3 \cdot 10^{-6}$
erreur globale : 1.18



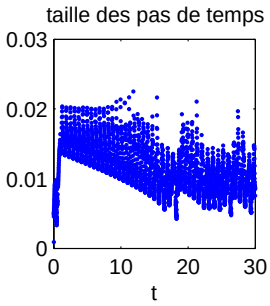
itération #2
nombre de pas de temps : 1087
évaluations pour calculer $\bar{X}$: 6643(10'898)
borne pour l'erreur locale ε : $1/3 \cdot 10^{-7}$
erreur globale : 8.75



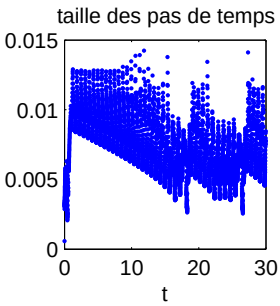
itération #3
nombre de pas de temps : 1656
évaluations pour calculer $\bar{X}$: 9979(20'877)
borne pour l'erreur locale ε : $1/3 \cdot 10^{-8}$
erreur globale : 2.73



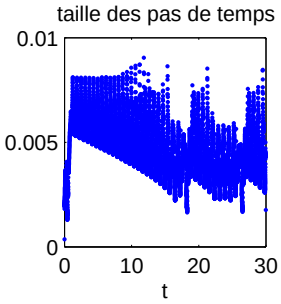
itération #4
nombre de pas de temps : 2696
évaluations pour calculer $\bar{X}$: 16'189(37'066)
borne pour l'erreur locale ε : $1/3 \cdot 10^{-9}$
erreur globale : 17.48



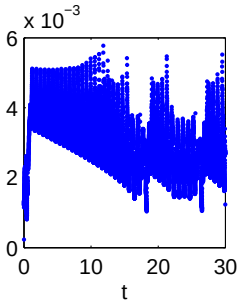
itération #5
nombre de pas de temps : 4173
évaluations pour calculer $\bar{X}$: 25'051(62'117)
borne pour l'erreur locale ε : $1/3 \cdot 10^{-10}$
erreur globale : 0.12



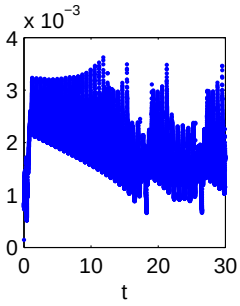
itération #6
nombre de pas de temps : 6637
évaluations pour calculer $\bar{X}$: 39'823(101'940)
borne pour l'erreur locale ε : $1/3 \cdot 10^{-11}$
erreur globale : 0.13



itération #7
nombre de pas de temps : 10'513
évaluations pour calculer $\bar{X}$: 63'085(165'025)
borne pour l'erreur locale ε : $1/3 \cdot 10^{-12}$
erreur globale : 0.015



itération #8
nombre de pas de temps : 16'651
évaluations pour calculer $\bar{X}$: 99'907(264'932)
borne pour l'erreur locale ε : $1/3 \cdot 10^{-13}$
erreur globale : 0.0015



5.3.3 Commentaires

On constate que le nombre de pas de temps pour la méthode `RK45` est multiplié à chaque fois par $1.6 \approx \sqrt[5]{10}$. C'est tout à fait logique, car 10 est le facteur par lequel on réduit la tolérance pour l'erreur locale et 5 est l'ordre de la méthode+1. Donc on constate que la méthode `RK45` ne fait que diviser la taille des pas de temps par $\sqrt[5]{10}$. Elle n'a en effet aucune autre information. Le comportement de l'erreur globale est très aléatoire. En effet, comme la méthode raffine partout au lieu de raffiner là où il faut, on n'est pas garanti de diminuer l'erreur à chaque itération. En fait, la méthode s'arrête quand la taille des premiers pas de temps (là où il faut être précis) est du même ordre de grandeur que pour `mstz`. On en conclut donc que la méthode `RK45` est tout à fait inappropriée à la résolution d'un problème chaotique.

Un des inconvénients de la méthode `mstz` est qu'il faut avoir sous la main la fonction $(a')^*$, ce qui n'est pas toujours évident dans la pratique. Néanmoins, on peut toujours la calculer numériquement, mais si le nombre de variables est grand cela risque de coûter très cher.

Un autre inconvénient de `mstz` est qu'il demande plus de mémoire. En effet, une méthode qui ne voit que l'erreur locale ne se préoccupe pas de la suite ni du passé. Seule l'erreur au pas de temps qu'elle est en train de calculer est importante. Par contre `mstz` a une vue sur l'ensemble de la solution. Pour calculer le dual, il faut avoir une estimation de X pour tous les pas de temps. De plus il faut garder en mémoire l'estimation de toutes les erreurs locales afin de calculer l'estimation

de l'erreur globale. Ceci peut causer des problèmes si la dimension de la solution est très grande ou alors si le nombre de pas de temps est très grand.

Chapitre 6

Application aux EDP

Dans ce chapitre, nous allons utiliser l'algorithme `mstz` pour résoudre la semi-discrétisation d'équations aux dérivées partielles (EDP). Notre but est d'étudier le comportement de l'algorithme sur certains problèmes types, comme l'équation de la chaleur et l'équation de Burgers. A quels endroits l'algorithme va-t-il raffiner ? Pour quels problèmes doit-on plus raffiner par rapport à d'autres ? C'est le type de question auxquelles nous répondrons dans ce chapitre.

6.1 Un peu de théorie

Tout d'abord, voici quelques notions de base liées à la résolution des EDP. Nous aurons besoin de ces notions par la suite. Les informations ci-dessous sont tirées notamment de [LeV90] et [QSS00].

6.1.1 Exemples et classification

Tout d'abord, voici 4 exemples d'EDP.

Exemple 6.1.1 (Poisson)

$$\begin{cases} -\dfrac{\partial^2}{\partial x^2}u(x,y) - \dfrac{\partial^2}{\partial y^2}u(x,y) = f(x,y) & \text{dans } \Omega \\ u(x,y) = 0 & \text{sur } \partial\Omega \end{cases}$$

avec Ω un domaine de $\mathbb{R}^2$ borné. Plongeons Ω dans l'espace (x,y,z) et supposons que Ω est inclus dans le plan (x,y) (le plan horizontal). On fixe les extrémités d'une membrane élastique sur $\partial\Omega$. $u(x,y)$ représente le déplacement selon z (i.e. verticalement) à la position (x,y) si l'on exerce une force verticale $f(x,y)$.

Exemple 6.1.2 (Chaleur)

$$\begin{cases} \dfrac{\partial}{\partial t}u(t,x) - \nu\dfrac{\partial^2}{\partial x^2}u(t,x) = f(t,x) & 0 \le x \le 1 \quad t \ge 0 \\ u(t,0) = u(t,1) = 0 & t \ge 0 \\ u(0,x) = u_0(x) & 0 \le x \le 1 \end{cases}$$

$u(t,x)$ décrit la température à un point x et au temps t d'une barre métallique de longueur 1 sous les conditions suivantes. Sa conductivité thermique est constante et égale à $\nu > 0$, ses extrémités

sont gardées à une température de 0 degrés. Au temps initial $t = 0$, sa température au point x est décrite par $u_0(x)$ et $f(t, x)$ représente la production de chaleur fournie au point x et au temps t. Ici, nous supposons que la densité volumétrique et la chaleur spécifique sont toutes deux constantes et unitaires.

Exemple 6.1.3 (Equation d'onde)

$$\begin{cases} \frac{\partial^2}{\partial t^2}u(t,x) - \gamma^2 \frac{\partial^2}{\partial x^2}u(t,x) = f(t,x) & 0 \leq x \leq 1 \quad t \geq 0 \\ u(0,x) = u_0(x) & 0 \leq x \leq 1 \\ \frac{\partial}{\partial t}u(0,x) = v_0(x) & 0 \leq x \leq 1 \end{cases}$$

$u(t, x)$ représente le déplacement transversal d'une corde élastique de longueur 1, fixée aux extrémités. γ est un coefficient dépendant de la masse spécifique de la corde et de sa tension. La corde est soumise à une force verticale $f(t, x)$. Les fonctions $u_0(x)$ et $v_0(x)$ représentent respectivement le déplacement initial et la vitesse initiale de la corde.

Exemple 6.1.4 (Burgers)

$$\begin{cases} \frac{\partial}{\partial t}u(t,x) + \frac{\partial}{\partial x}\frac{u^2(t,x)}{2} = 0 & 0 < x < 1 \quad t > 0 \\ u(0,x) = u_0(x) & 0 < x < 1 \\ u(t,0) = u_0(0) \geq 0 & t > 0 \\ \frac{\partial}{\partial x}u(t,1) = 0 & t > 0 \end{cases}$$

Cette équation est un modèle très simplifié de la dynamique des fluides.

L'ordre d'une EDP est sa plus haute dérivée partielle. Une EDP est dite linéaire si les termes qui multiplient les dérivées partielles sont indépendants de u. Les trois premières équations sont des équations linéaires d'ordre 2. La dernière équation est non-linéaire d'ordre 1. On peut la réécrire

$$\frac{\partial}{\partial t}u(t,x) + u(t,x)\frac{\partial}{\partial x}u(t,x) = 0$$

Pour les équations d'ordre 2, si l'on remplace les termes de type $\frac{\partial^2}{\partial x^2}$ par x^2, ceux de type $\frac{\partial}{\partial t}$ par t et f par -1, l'on obtient :

Poisson : $x^2 + y^2 = 1 \Rightarrow$ l'équation est appelée **elliptique**

Chaleur : $-t + \nu x^2 = 1 \Rightarrow$ l'équation est appelée **parabolique**

Onde : $-t^2 + \gamma^2 x^2 = 1 \Rightarrow$ l'équation est appelée **hyperbolique**

Il existe une classification similaire des équations du premier degré, mais nous n'entrerons pas ici dans les détails. Toutefois, si l'on a affaire à une seule équation aux dérivées partielles de degré 1 et non un système d'EDP, alors l'EDP est de type hyperbolique. De cette manière, l'équation de Burgers est une équation hyperbolique.

6.1.2 Réduire une EDP à une EDO : la semi-discrétisation

Nous nous concentrons sur les équations paraboliques de degré 2 ainsi que sur les équations de degré 1. C'est le type d'équations qui se ramènent le plus facilement à une EDO. Pour résoudre

une EDP numériquement, il faut discrétiser toutes les variables. Supposons qu'on ait une variable temporelle t et une variable spatiale unidimensionnelle x. On peut commencer par discrétiser en espace. Plusieurs possibilités s'offrent à nous : différences finies, éléments finis, méthodes spectrales, ... Dans le cas des différences finies, on considère un maillage $x_0 < x_1 < \ldots < x_M$ et on se concentre sur l'évolution de la fonction $u(t,x)$ en ces points. Autrement dit, nos inconnues sont les $M+1$ fonctions $u(t,x_0), u(t,x_1), \ldots, u(t,x_M)$. En approximant les dérivées numériquement, on obtient un système de $M+1$ EDO (voir l'exemple de la chaleur à la section 6.2). Cette démarche est appelée semi-discrétisation d'une EDP. Si l'on emploie d'autres méthodes comme les éléments finis ou les méthodes spectrales (cf chapitre 7) on obtient aussi des systèmes d'EDO. C'est ces systèmes que nous allons résoudre à l'aide de l'algorithme `mstz`.

En particulier, il y a deux types d'erreurs. Considérons que la semi-discrétisation s'est faite par différences finies à l'aide d'un maillage uniforme $x_0 < x_1 < \ldots < x_M$ dont la taille du maillage est h (i.e. $x_j - x_{j-1} = h \ \ \forall j$). On note :

$$\mathbf{u}(t) = \begin{pmatrix} u(t,x_0) \\ u(t,x_1) \\ \\ \cdots \\ \\ u(t,x_M) \end{pmatrix}$$

La semi-discrétisation aboutit à un système d'EDO qui a la forme suivante :

$$\begin{cases} \dot{\mathbf{u}}^{\mathbf{h}}(t) = a(t, \mathbf{u}^{\mathbf{h}}(t)) & \forall t \in I \\ \mathbf{u}^{\mathbf{h}}(0) = \mathbf{u_0} \end{cases} \tag{6.1}$$

Avec $\mathbf{u_0}$ la discrétisation des conditions initiales de l'EDP. On a que $u_j^h(t) \approx u(t,x_j)$ où $u(t,x)$ est la solution de l'EDP.

En résolvant le système d'EDO (6.1) on approximera la solution exacte du système semi-discrétisé $\mathbf{u_h}(t)$ par $\bar{\mathbf{u}}_{\mathbf{h}}(t)$. Soit $g : \mathbb{R}^{M+1} \to \mathbb{R}$, et T le temps final. Comme dans les chapitres précédents, nous voulons minimiser $g(\mathbf{u}(T)) - g(\bar{\mathbf{u}}^{\mathbf{h}}(T))$. Nous noterons :

- $\mathcal{E}$, l'erreur totale $g(\mathbf{u}(T)) - g(\bar{\mathbf{u}}^{\mathbf{h}}(T))$
- $\mathcal{E}^h$, l'erreur spatiale $g(\mathbf{u}(T)) - g(\mathbf{u}^{\mathbf{h}}(T))$
- $\mathcal{E}^t$, l'erreur temporelle $g(\mathbf{u}^{\mathbf{h}}(T)) - g(\bar{\mathbf{u}}_{\mathbf{h}}(T))$

Nous avons bien sûr $\mathcal{E} = \mathcal{E}^h + \mathcal{E}^t$. Les estimations de ces erreurs seront notées $\overline{\mathcal{E}}$, $\overline{\mathcal{E}}^h$, et $\overline{\mathcal{E}}^t$. L'estimation d'erreur fournie par `mstz` ne concerne que $\mathcal{E}^t$, l'erreur temporelle et donc seule cette erreur sera sous la tolérance TOL. L'erreur spatiale dépend du schéma que nous employons pour semi-discrétiser l'EDP.

6.1.3 Analyse de stabilité

La stabilité d'une méthode numérique pour la résolution d'une équation aux dérivées partielles dépend du type de problème. Les notions sont différentes si l'on a affaire à un problème elliptique, hyperbolique ou parabolique. Néanmoins, les concepts de base sont les mêmes que ceux énoncés lors de la discussion de la stabilité des EDO à la section 2.2. Comme le but de ce travail n'est point de faire une étude de stabilité, nous nous contenterons de mentionner, pour chaque problème que nous traiterons, quelles sont les contraintes assurant la stabilité du schéma numérique que nous utilisons.

6.2 Equation de la chaleur

Comme premier cas test de l'algorithme `mstz` utilisé pour résoudre des EDP, nous allons résoudre la semi-discrétisation par différences finies de l'équation de la chaleur.

$$\begin{cases} \dfrac{\partial}{\partial t}u(t,x) = \nu\dfrac{\partial^2}{\partial x^2}u(t,x) & 0 \leq x \leq 1 \quad 0 \leq t \leq 1 \\ u(t,0) = u(t,1) = 0 & 0 \leq t \leq 1 \\ u(0,x) = \sin(2\pi x) & 0 \leq x \leq 1 \end{cases} \tag{6.2}$$

Cette semi-discrétisation s'effectue de la manière suivante. On prend un maillage uniforme en espace $0 = x_0 < \ldots < x_M = 1$ de pas de discrétisation h. On note $u_j(t) = u(t, x_j)$. En approximant la deuxième dérivée de la manière suivante :

$$\left.\frac{\partial^2}{\partial x^2}u(t,x)\right|_{x=x_j} \approx \frac{u_{j+1}(t) - 2u_j(t) + u_{j-1}(t)}{h^2} \quad \forall j = 1, \ldots, M-1 \tag{6.3}$$

on obtient, en notant $\mathbf{u^h}$ la solution du système semi-discrétisé,

$$\begin{aligned} \frac{d}{dt}u_j^h(t) &= \nu\frac{u_{j+1}^h(t) - 2u_j^h(t) + u_{j-1}^h(t)}{h^2} \quad \forall j = 1, \ldots, M-1 \\ u_0^h(t) = u_M^h(t) &= 0 \quad \forall t \\ u_j^h(0) &= u_0(x_j) \quad \forall j = 0, \ldots, M \end{aligned}$$

Sous forme matricielle, on a :

$$\begin{cases} \dot{\mathbf{u}}^{\mathbf{h}}(t) = \frac{\nu}{h^2}A\mathbf{u}^{\mathbf{h}}(t) & \forall t > 0 \\ \mathbf{u}^{\mathbf{h}}(0) = \mathbf{u_0} \end{cases} \tag{6.4}$$

avec $\mathbf{u}^{\mathbf{h}}(t) = (u_1^h(t), \ldots, u_{M-1}^h(t))^T$ le vecteurs des inconnues, $\mathbf{u_0} = (u_0(x_1), \ldots, u_0(x_{M-1}))^T$ et

$$A = \begin{pmatrix} -2 & 1 & 0 & . & . & . & 0 \\ 1 & -2 & 1 & 0 & . & . & 0 \\ 0 & 1 & -2 & 1 & 0 & . & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & 1 & -2 & 1 \\ 0 & . & . & . & . & 1 & -2 \end{pmatrix} \in \mathbb{R}^{(M-1)\times(M-1)}$$

On a donc un système d'EDO à $M-1$ inconnues que l'on peut résoudre avec la méthode `mstz`. L'avantage de l'équation de la chaleur (6.2) est que l'on peut connaître la solution exacte u ainsi que la solution exacte du système semi-discrétisé $\mathbf{u^h}$. On trouve la solution exacte avec les séries de Fourier (cf [QSS00]). Dans notre cas, elle vaut :

$$u(t,x) = \mathrm{e}^{-4\nu\pi^2 t}\sin(2\pi x) \tag{6.5}$$

La solution du système semi-discrétisé est :

$$\mathbf{u}^{\mathbf{h}}(t) = \mathrm{e}^{\frac{\nu}{h^2}At}\mathbf{u_0}$$

où l'on prend l'exponentielle d'une matrice. Comme méthode locale pour l'avancement en temps, nous prendrons la méthode d'Euler progressive. Etudions la stabilité de ce schéma numérique. Nous supposons maintenant que tous les pas de temps auront la même longueur Δt. On remarque en observant la solution exacte (6.5) que $\lim_{t\to\infty} u(t,x) = 0$. Logiquement la solution numérique devrait avoir le même comportement. Ceci est cohérent avec ce que nous avons fait pour les EDO à la section 2.2. Par la méthode d'Euler progressive, on a :

$$\begin{aligned}\bar{\mathbf{u}}^{\mathbf{h}}(t_{n+1}) &= \bar{\mathbf{u}}^{\mathbf{h}}(t_n) + \Delta t\frac{\nu}{h^2}A\bar{\mathbf{u}}^{\mathbf{h}}(t_n) = (I + \Delta t\frac{\nu}{h^2}A)\bar{\mathbf{u}}^{\mathbf{h}}(t_n)\\ &= (I + \Delta t\frac{\nu}{h^2}A)^{n+1}\mathbf{u_0}\end{aligned}$$

Donc $\bar{\mathbf{u}}^{\mathbf{h}}(t_n) \to 0$ quand $n \to \infty$ si et seulement les valeurs propres de $(I + \Delta t\frac{\nu}{h^2}A)$ ont toutes un module < 1. Ceci est vérifié si et seulement si

$$\Delta t < \frac{h^2}{2\nu} \qquad \text{(cf [QSS00])} \tag{6.6}$$

Dans nos tests, nous prendrons $h = 1/15$. Le Δt initial vaudra 0.01 et ν vaudra 0.1.

Pour la fonction g, nous choisissons un point de contrôle $\bar{x} \in [0,1]$ où nous voulons estimer l'erreur. On pourrait imaginer prendre $\bar{x} = x_j$ un point du maillage. On aurait alors $g(\mathbf{u}(T)) = u(T, x_j)$. Pour résoudre le dual, la condition initiale (qui se trouve à la fin de l'intervalle temporel) est $\psi(T, \mathbf{u}(T)) = \nabla g(\mathbf{u}(T)) = (0, \ldots, 0, 1, 0, \ldots, 0)$ dans notre cas, avec le 1 à la j-ème place. Ce n'est pas très lisse dans le domaine des x. C'est pourquoi, nous pondérerons le point $\bar{x}$ et ses voisins par une loi normale. Ainsi, $\nabla g(\mathbf{u}(T))$ n'aura pas qu'une seule composante non nulle :

$$g(\mathbf{u}(T)) = \sum_{j=0}^{M-1} w_j u_j(T) = \sum_{j=0}^{M-1} \frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{1}{2}\left(\frac{x_j-\bar{x}}{\sigma}\right)^2} u_j(T) \quad \text{avec } \sigma = 0.05 \tag{6.7}$$

$$(\nabla g(\mathbf{u}(T)))_j = w_j = \frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{1}{2}\left(\frac{x_j-\bar{x}}{\sigma}\right)^2} \tag{6.8}$$

Nous prendrons $\bar{x} = 0.4$. Les résultats sont dans la table 6.1 et dans la figure 6.1.

TOL	$\mathcal{E}$	$\mathcal{E}^h$	$\mathcal{E}^t$	$\overline{\mathcal{E}}^t$	# eval. a	# eval. $(a')^*$	CPU	N	it
0.001	-0.00876	-0.00957	0.000811	0.000809	4500	1500	2.14	800	4

TAB. 6.1 – Résultats pour l'équation de la chaleur

L'algorithme effectue un raffinement uniforme. Ce cas test est donc en quelque sorte l'analogue de l'exponentielle, mais pour les EDP. On remarque que l'estimation $\overline{\mathcal{E}}^t$ de l'erreur temporelle $\mathcal{E}^t$ est bonne. C'est l'erreur spatiale qui domine nettement dans l'erreur totale.

6.3 Equation de la chaleur avec blow-up

Pour pousser l'algorithme un peu plus loin, nous voulons voir comment il gère une situation de blow-up pour des EDP. Nous avons pris le problème de la chaleur et nous avons modifié les conditions

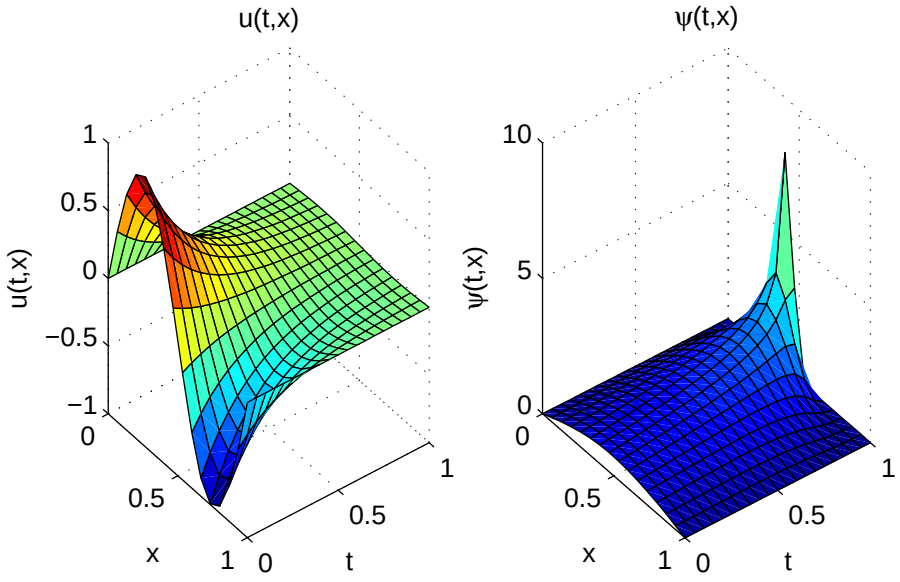


FIG. 6.1 – Les résultats pour l'équation de la chaleur et son dual

aux bords par des équations non-linéaires, suivant la procédure tirée de [ADR02]. Nous voulons voir où l'algorithme choisit de raffiner. Est-ce plutôt au début de l'intervalle temporel ou à la fin ?

$$\begin{cases} \dfrac{\partial}{\partial t}u(t,x) = \dfrac{\partial^2}{\partial x^2}u(t,x) & 0 \le x \le 1 \quad 0 \le t \le 0.18 \\ \dfrac{\partial}{\partial x}u(t,0) = 0 & 0 \le t \le 1 \\ \dfrac{\partial}{\partial x}u(t,1) = u^2(t,1) & 0 \le t \le 1 \\ u(0,x) = 1 & 0 \le x \le 1 \end{cases}$$

La valeur finale de l'intervalle de temps, 0.18, est prise de manière à éviter que les valeurs des fonctions soient trop grandes. Comme précédemment, nous définissons un maillage uniforme en espace $0 = x_0 < \ldots < x_M = 1$. Le pas de discrétisation est h. Cette fois nous avons 2 inconnues de plus qu'avant : $u_0^h(t)$ et $u_M^h(t)$. Avant elles valaient zéro. Les variables $u_j(t)$ pour $1 \le j \le M-1$ sont traitées de la même manière qu'auparavant en employant les différences centrées de deuxième ordre (6.3). Pour nos deux nouvelles variables, voici la procédure. On note

$$u_{-1}^h \approx u(t,-h) \text{ et } u_{M+1}^h \approx u(t,1+h)$$

$$\begin{aligned} \text{Utilisant } u(t,x+h) &\approx u(t,x-h) + 2h\frac{\partial u(t,x)}{\partial x} \\ \text{On a } u_{M+1}^h(t) &\approx u_{M-1}^h(t) + 2h\underbrace{\frac{\partial u_M^h(t)}{\partial x}}_{\left(u_M^h(t)\right)^2} = u_{M-1}^h(t) + 2h(u_M^h(t))^2 \\ \text{et } u_{-1}^h(t) &\approx u_1^h(t) - 2h\underbrace{\frac{\partial u_0^h(t)}{\partial x}}_{0} = u_1^h(t) \end{aligned}$$

$$\begin{aligned}
\text{Avec (6.3), on obtient } \dot{u}_M^h(t) &= \frac{u_{M+1}^h(t) - 2u_M^h(t) + u_{M-1}^h(t)}{h^2} \\
&= \frac{2u_{M-1}^h(t) - 2u_M^h(t)}{h^2} + \frac{2}{h}(u_M^h(t))^2 \\
\text{et } \dot{u}_0^h(t) &= \frac{u_1^h(t) - 2u_0^h(t) + u_{-1}^h(t)}{h^2} = \frac{2u_1^h(t) - 2u_0^h(t)}{h^2}
\end{aligned}$$

En version matricielle, cela donne :

$$\begin{cases} \dot{\mathbf{u}}^{\mathbf{h}}(t) = \frac{1}{h^2}\hat{A}\mathbf{u}^{\mathbf{h}}(t) + \frac{2(u_M^h(t))^2}{h}\mathbf{e_M} & \forall t > 0 \\ \mathbf{u}(0) = \mathbf{u_0} \end{cases}$$

avec

$$\begin{aligned}
\mathbf{u}^{\mathbf{h}}(t) &= (u_0(t), u_1(t), \ldots, u_{M-1}(t), u_M(t))^T \text{ le vecteurs des inconnues} \\
\mathbf{u_0} &= (1 \ldots, 1)^T \\
\mathbf{e_M} &= (0, \ldots, 0, 1)^T
\end{aligned}$$

$$\hat{A} = \begin{pmatrix} -2 & 2 & 0 & . & . & . & 0 \\ 1 & -2 & 1 & 0 & . & . & 0 \\ 0 & 1 & -2 & 1 & 0 & . & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & 1 & -2 & 1 \\ 0 & . & . & . & . & 2 & -2 \end{pmatrix} \in \mathbb{R}^{(M+1)\times(M+1)}$$

Nous prenons la même fonction g que précédemment mais avec $\bar{x} = 1$, car c'est cet endroit qui nous intéresse, car c'est là qu'il y a la condition aux bords non-linéaire. On prend $h = 0.05$ et $\Delta t = 0.002$. La méthode locale employée sera Runge-Kutta. On obtient les résultats de la table 6.2 et de la figure 6.2.

TOL	$\mathcal{E}$	$\mathcal{E}^h$	$\mathcal{E}^t$	$\overline{\mathcal{E}}^t$	# eval. a	# eval. $(a')^*$	CPU	N	it
1e-05	-	-	-	-3.52e-07	5814	1938	1.76	123	3

TAB. 6.2 – Résultats pour l'équation de la chaleur avec blowup

Le dual indique donc qu'il faut raffiner près du blow-up, résultat attendu par Zunino et Calabrò en relation avec leur travail pour [ZC04]. La partie en bas à droite de la figure 6.2 montre les valeurs du dual au cours du temps pour le point $x = 1$ où est imposé la condition non-linéaire. Le dual détecte vraiment la fin comme l'endroit où il faut raffiner. En effet, proche de la fin, il grandit de manière très raide puis redescend pour rejoindre les conditions initiales qui lui sont fixées.

6.4 Equation de Burgers

Maintenant, nous allons employer le code pour résoudre une équation aux dérivées partielles non-linéaire, avec des conditions initiales discontinues. Grâce à l'estimation a posteriori de l'erreur,

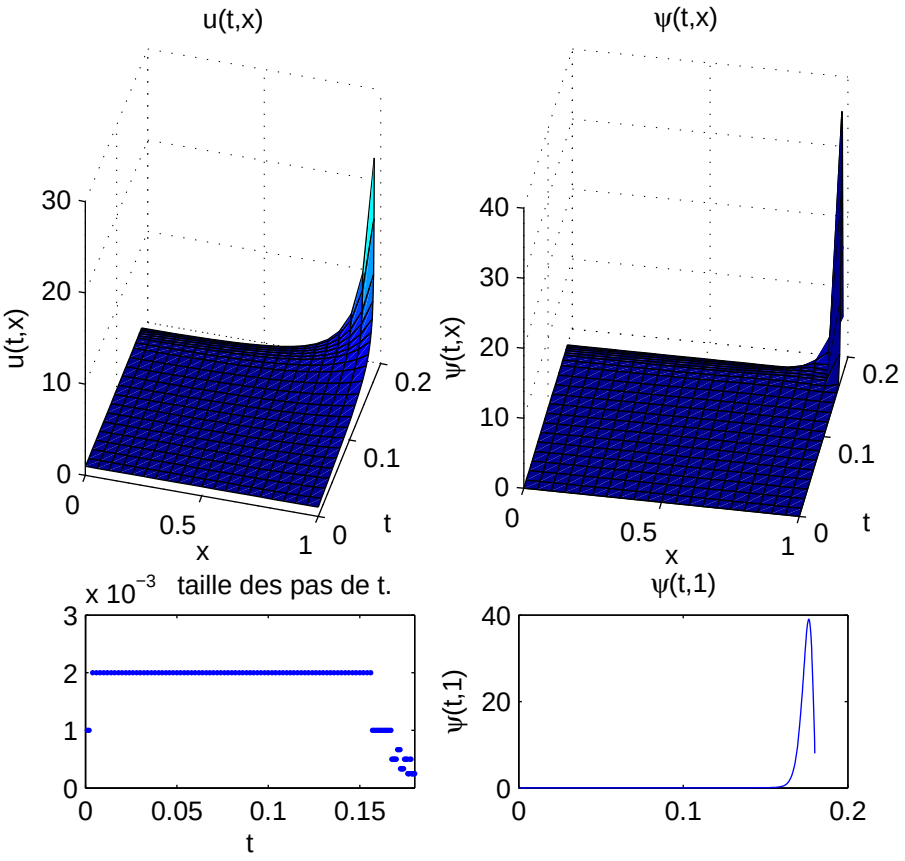


FIG. 6.2 – Les résultats pour l'équation de la chaleur avec blow-up

nous pouvons étudier quelques propriétés du problème. Nous étudierons quelle type de conditions initiales sont plus difficiles à résoudre.

Tout comme le modèle vu à la section 4.3.5, l'équation de Burgers est une simplification du modèle de Navier-Stokes qui gouverne toute la dynamique des fluides.

$$\begin{cases} \dfrac{\partial}{\partial t}u(t,x) + \dfrac{\partial}{\partial x}\dfrac{u^2(t,x)}{2} = 0 & -1 \le x \le 1 \quad 0 \le t \le 1 \\ u(0,x) = u_0(x) & -1 \le x \le 1 \\ \dfrac{\partial}{\partial x}u(t,0) = 0 & 0 \le t \le 1 \\ \dfrac{\partial}{\partial x}u(t,1) = 0 & 0 \le t \le 1 \end{cases}$$

L'équation de Burgers est une équation hyperbolique de degré 1. Le problème des équations hyperboliques est que des données initiales continues peuvent donner des solutions discontinues qui sont très difficiles à appréhender numériquement. Pour résoudre ce problème, on ajoute un terme (dit "de viscosité") à l'équation de Burgers pour la rendre parabolique. Ainsi, l'on est garanti que les solutions seront toujours continues pour $t > 0$ et ceci même si les conditions initiales sont discontinues (cf. [LeV90]). Voici cette modification (sur le membre de droite) :

$$\frac{\partial}{\partial t}u(t,x) + \frac{\partial}{\partial x}\frac{u^2(t,x)}{2} = \varepsilon\frac{\partial^2}{\partial x^2}u(t,x) \tag{6.9}$$

Cette équation est appelée équation de Burgers visqueuse.

Pour la discrétisation en espace, nous avons utilisé des schémas centrés d'ordre 2. On emploie à nouveau un maillage uniforme $-1 = x_0 < \ldots < x_M = 1$ avec comme taille de la maille h. Voici ces schémas :

$$\begin{aligned}
\frac{\partial}{\partial x}\frac{u^2(t,x)}{2}\bigg|_{x=x_j} &\approx \frac{u_{j+1}^2(t) - u_{j-1}^2(t)}{4h} \quad \forall j = 1,\ldots,M-1 \\
\frac{\partial^2}{\partial x^2}u(t,x)\bigg|_{x=x_j} &\approx \frac{u_{j+1}(t) - 2u_j(t) + u_{j-1}(t)}{h^2} \quad \forall j = 1,\ldots,M-1
\end{aligned}$$

Cela donne le système suivant :

$$\begin{aligned}
\dot{u}_j^h(t) &= -\frac{\left(u_{j+1}^h(t)\right)^2 - \left(u_{j-1}^h(t)\right)^2}{4h} + \varepsilon\frac{u_{j+1}^h(t) - 2u_j^h(t) + u_{j-1}^h(t)}{h^2} \quad \forall j = 1,\ldots,M-1 \\
\dot{u}_0^h(t) &= \dot{u}_1^h(t) \\
\dot{u}_M^h(t) &= \dot{u}_{M-1}^h(t)
\end{aligned}$$

Nous avons considéré deux types de conditions initiales, l'escalier descendant et montant, comme on peut le voir sur la figure 6.3.

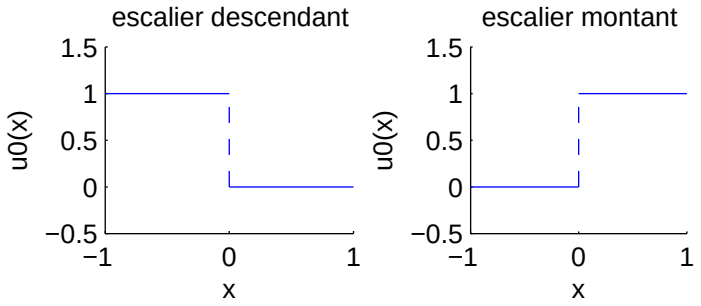


FIG. 6.3 – Les conditions initiales

Il est difficile d'étudier la stabilité des schémas numériques pour les systèmes non-linéaires. En effet, dépendant des valeurs de h, Δt et ε, les schémas peuvent devenir instables, c'est-à-dire que des oscillations numériques importantes apparaissent et qu'elles dominent la solution. Néanmoins, quelques critères donnent certaines conditions. Par exemple, quand $\varepsilon = 0$, i.e. l'équation est hyperbolique, on a la définition et le critère suivant [QSS00] :

Definition 6.4.1 *Une méthode numérique (avec un maillage uniforme en temps et en espace de mailles Δt et h) pour un problème hyperbolique (linéaire ou non-linéaire) est dit stable si, pour n'importe quelle valeur T, s'il existe deux constantes $C_T > 0$ et $\delta_0 > 0$, t.q.*

$$\|\bar{\mathbf{u}}^{\mathbf{h}}(t_n)\|_\Delta \leq C_T\|\bar{\mathbf{u}}^{\mathbf{h}}(t_0)\|_\Delta$$

pour tous les n t.q. $n\Delta t \leq T$ et tous les Δt, h t.q $0 < \Delta t \leq \delta_0$ et $0 < h \leq \delta_0$. $\|\cdot\|$ est une norme discrète, comme par exemple :

$$\|v\|_\Delta = \left(h \sum_{j=0}^{M} |v_j|^2 \right)^{\frac{1}{2}}$$

Propriété 6.4.2 (condition CFL) *Soit $w(t,x)$ une fonction donnée et x de dimension 1. Pour qu'une méthode numérique explicite pour résoudre un système hyperbolique du type*

$$\frac{\partial}{\partial t}u(t,x) + w(t,x)\frac{\partial}{\partial x}u(t,x) = 0$$

soit stable, il est nécessaire et suffisant que

$$\Delta t \leq \frac{h}{\sup_{t,x} |w(t,x)|} \tag{6.10}$$

Remarque 6.4.3 *En fait, la condition (6.10) n'est pas valable pour toutes les méthodes numériques mais seulement pour un sous-ensemble de méthodes ayant certaines propriétés. Néanmoins, toutes les méthodes que nous emploierons font partie de ce sous-ensemble, donc nous ne détaillerons pas les conditions ici. Le lecteur peut se référer à [QSS00].*

Dans notre cas, la fonction $w(t,x)$ est $u(t,x)$ qui est inconnue. On prendra donc $\sup_x |u(0,x)| = 1$ comme approximation de $\sup_{t,x} |u(t,x)|$. Ainsi la condition CFL devient

$$\Delta t \leq h \tag{6.11}$$

Le terme visqueux que l'on a ajouté est identique à celui de l'équation de la chaleur. Donc, comme on va à nouveau employer la méthode d'Euler progressive pour avancer en temps, on peut s'attendre à avoir le même type de condition de stabilité que (6.6) :

$$\Delta t < \frac{h^2}{2\varepsilon} \tag{6.12}$$

Donc en respectant les conditions (6.11) et (6.12), le schéma devrait avoir de bonnes propriétés de stabilité. C'est ce que nous avons observé dans nos expériences. Néanmoins, il y a des cas où ces conditions sont respectées mais le schéma est quand même instable.

Nous avons pris les valeurs suivantes : $\varepsilon = 0.01$, $h = 0.025$, $\Delta t = 0.005$. Nous contrôlons l'erreur au point $\bar{x} = 0.5$ et employons la même fonction g qu'auparavant (cf. (6.7)). Nous prendrons $\sigma = 0.1$ car le domaine des x est 2 fois plus long que pour l'équation de la chaleur. Les solutions exactes pour ce problème sont connues. Pour le cas de l'escalier montant, on a la solution suivante :

$$u(t,x) = \frac{1}{2}(1 - \tanh(\frac{x - 0.5t}{4\varepsilon}))$$

Quand $\varepsilon \to 0$, la solution consiste simplement à un escalier montant se déplaçant vers la droite avec une vitesse 0.5. Cette solution est appelée onde de choc. Pour l'escalier descendant voici la solution qu'on obtient lorsque on fait tendre ε vers 0 :

$$u(x,t) = \begin{cases} 0 & x < 0 \\ x/t & 0 \leq x \leq t \\ 1 & x > t \end{cases}$$

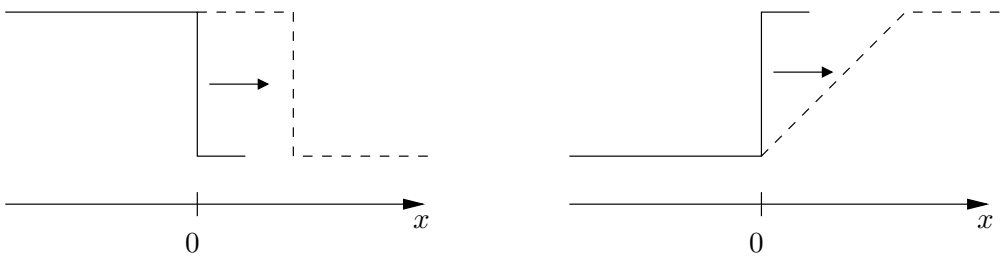


FIG. 6.4 – Les solutions : onde de choc et onde de raréfaction

Cette solution est appelée onde de raréfaction. On peut voir ces deux solutions dans la figure 6.4.

On peut voir les résultats des simulations dans les figures 6.6 et 6.7. On remarque que dans le cas de l'escalier montant, l'algorithme raffine plus au début, alors que dans le cas de l'escalier descendant c'est le contraire.

Pour l'escalier descendant, la raison numérique d'un tel raffinement est la suivante. Comme l'erreur locale a une partie positive et une partie négative, la somme des erreurs locales est environ zéro. Au début de l'intervalle de temps, le dual est à peu près constant pour toute la partie où l'erreur locale est différente de zéro. Donc $\left(\bar{e}(t_i), \bar{\psi}(t_i)\right) \approx \sum_{j=1}^{d} e_j(t_n) \approx 0$. Par contre, quand t est proche de 1, le dual est positif quand l'erreur est dans sa partie négative, et le dual est nul quand l'erreur est dans sa partie positive. Donc $\left(\bar{e}(t_i), \bar{\psi}(t_i)\right)$ est en quelque sorte la somme de toutes les composantes de $\bar{e}(t_i)$ qui sont négatives.

Pour l'escalier montant, la raison numérique du raffinement pour les petits t est que l'erreur locale est nettement plus grande au début de l'intervalle temporel qu'à la fin.

On remarque que le problème de l'escalier montant est plus difficile à résoudre que celui de l'escalier descendant. En effet, pour les mêmes paramètres, il faut seulement 3168 évaluations de la fonction a pour trouver l'onde de choc alors qu'il en faut 48'750 pour résoudre l'onde de raréfaction. En soit, ceci est très étonnant mais confirme le résultat de la première partie de [Bur98]. En effet, on pourrait penser que l'onde de choc et sa pente très raide est plus difficile à appréhender que l'onde de raréfaction.

Il n'en est pas le cas pour la raison suivante. Dans l'analyse des équations hyperboliques, un concept très utile est celui des *caractéristiques*. Ce sont des courbes dans le plan $x-t$ qui indiquent en quelque sorte, comment les ondes se propagent au cours du temps. Par exemple, si le problème est linéaire, la valeur de $u(t,x)$ est constante le long des caractéristiques. Leur définition exacte peut être trouvée dans le chapitre 3 de [LeV90]. Dans le cas de l'équation de Burgers, la pente de la caractéristique au point (x_0, t_0) est égale à la valeur de $\frac{1}{u(t_0,x_0)}$. Ceci est illustré dans la figure 6.5.

Dans le cas de l'onde de choc (fonction décroissante, à gauche), les zones influencées par une perturbation (le rectangle noir) décroissent avec le temps. En particulier, lorsque les caractéristiques se croisent, ces perturbations sont comme oubliées, absorbées par la collision des caractéristiques. Par contre dans le cas de l'onde de raréfaction (fonction croissante, à droite), ces perturbations se propagent sur une région de plus en plus importante. Ceci explique les deux phénomènes observées :

1. L'onde de raréfaction est beaucoup plus chère à calculer que l'onde de choc. En effet, comme les perturbations s'amplifient, il faut faire des calculs précis pour ne pas avoir d'amplification des erreurs.

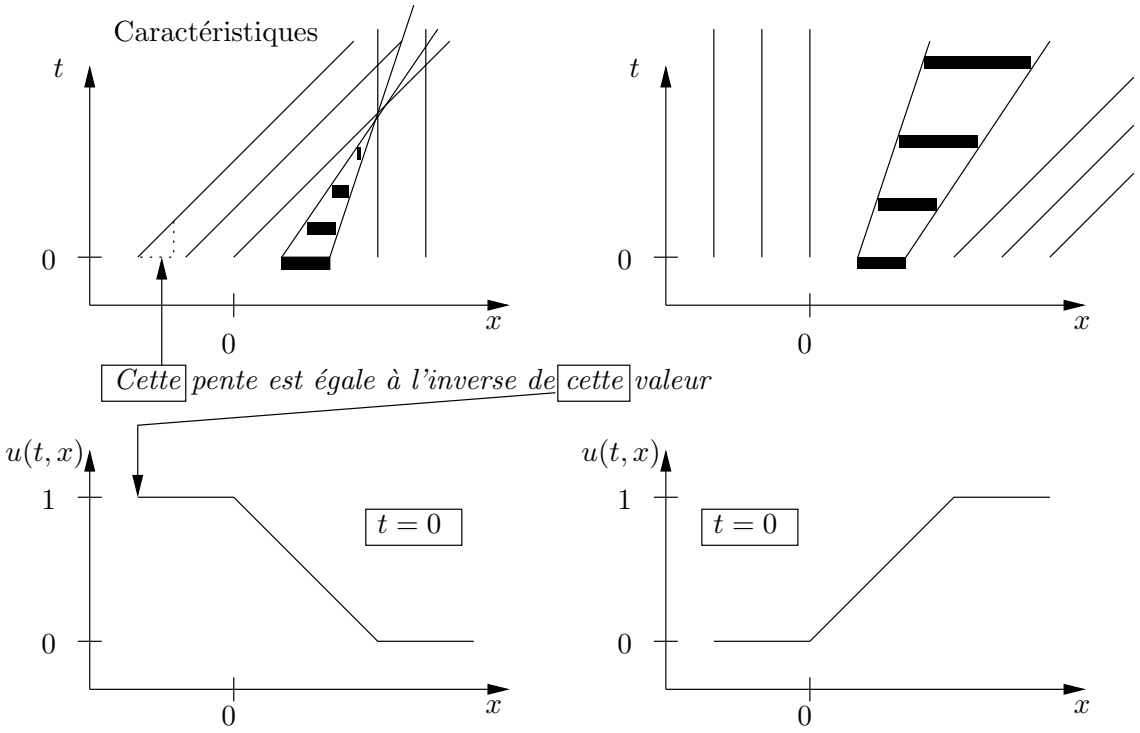


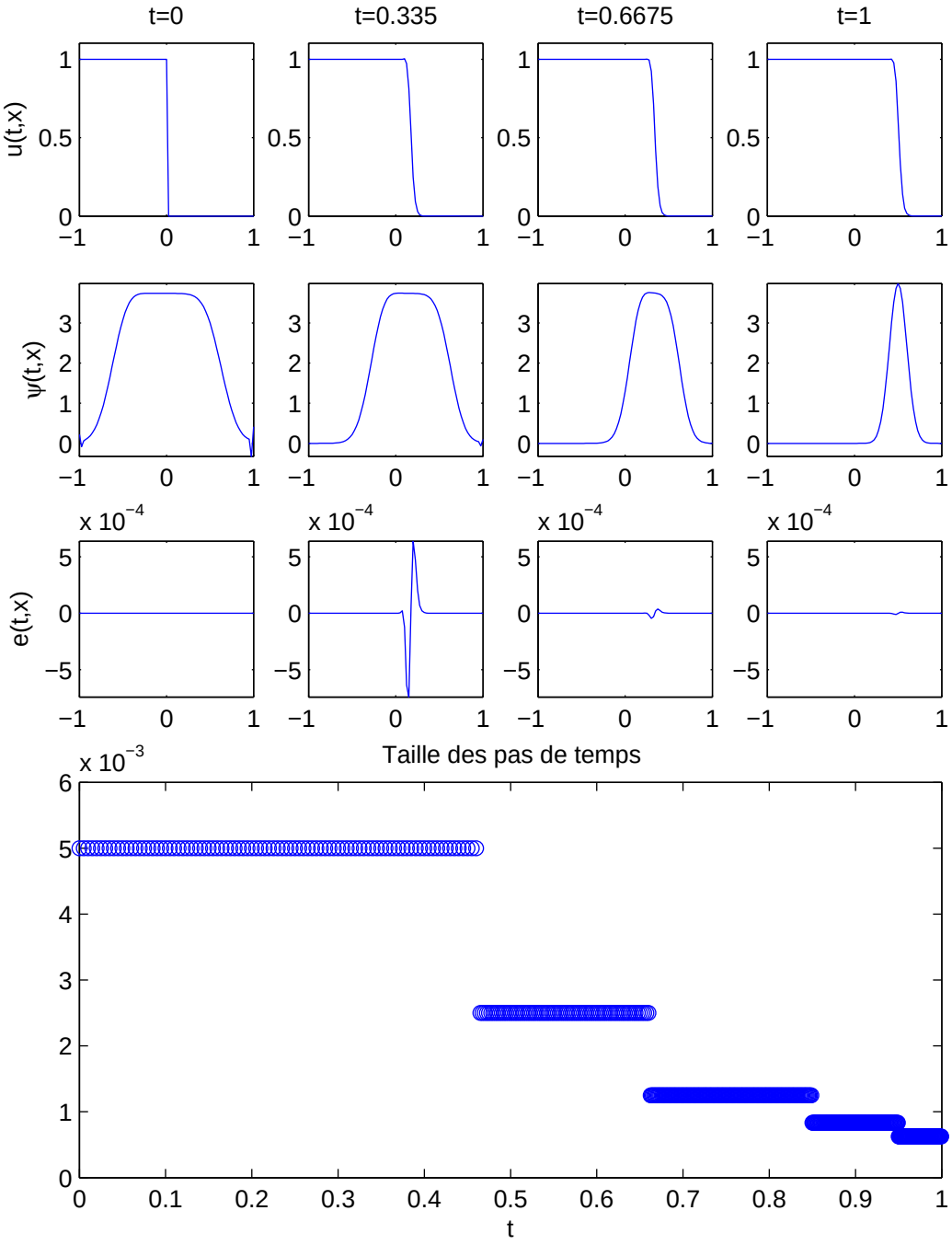
FIG. 6.5 – Les lignes caractéristiques pour le problème de Burgers. A gauche, un comportement similaire à une onde de choc. A droite celui d'une onde de raréfaction. Les rectangles noirs représentent l'évolution d'une région au cours du temps. Par exemple, si une perturbation est créée dans le rectangle noir, les endroits où on l'observera plus tard sont aussi dans ce rectangle noir. On remarque que lorsque les caractéristiques se croisent, le rectangle noir disparaît.

2. Il faut raffiner au début pour l'onde de raréfaction. En effet, si une erreur est commise au début, elle sera amplifiée tout au long de l'intervalle temporel. Par contre, pour l'onde de choc, une erreur au début va être atténuée au fil du temps. Donc il est seulement important de faire des calculs précis vers la fin de l'intervalle temporel, proche de l'endroit où nous calculons l'erreur globale.

Remarque 6.4.4 *Les explications ci-dessus sur l'amplification ou l'amortissement des perturbations ne sont valables que si les perturbations ont une masse nulle. Autrement dit, si u est la solution non perturbée et v la solution perturbée pour un instant t donné, alors il faut que*

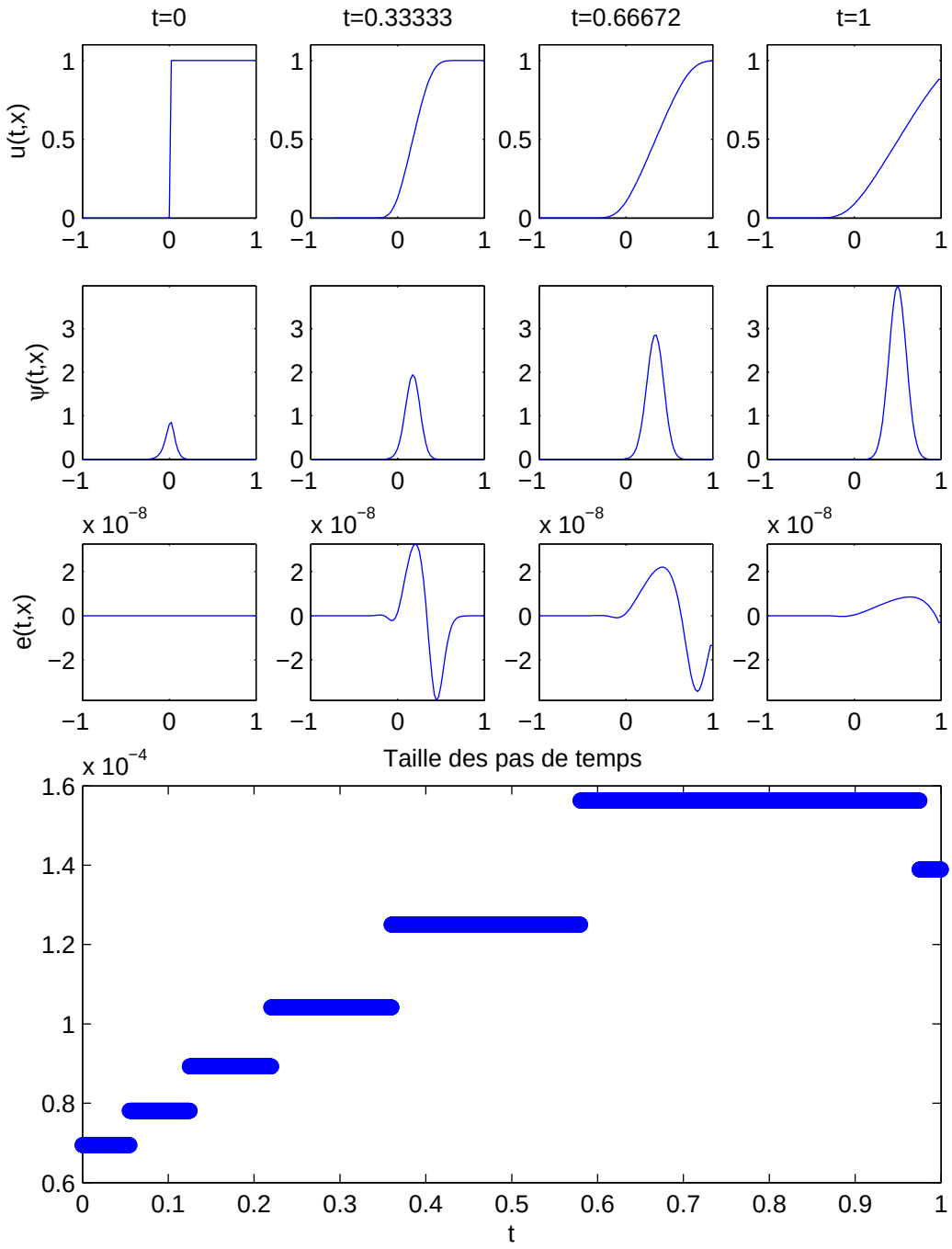
$$\int_{-1}^{1} (u(t,x) - v(t,x))dx = 0 \tag{6.13}$$

Néanmoins, il est connu que le schéma centré que nous avons employé est conservatif, c'est-à-dire que toutes les perturbations numériques qu'il introduit ont la propriété 6.13.



	TOL	$\overline{\mathcal{E}}^t$	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	0.005	-0.00354	3168	1056	2.05	523	3

FIG. 6.6 – Problème de Burgers dans le cas de l'onde de choc



	TOL	$\overline{\mathcal{E}}^t$	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	0.005	0.00365	48750	16250	37.34	8565	5

FIG. 6.7 – Problème de Burgers dans le cas de l'onde de raréfaction

Chapitre 7

L'équation de Kuramoto-Sivashinsky

Dans ce dernier chapitre, nous allons étudier une équation aux dérivées partielles chaotique du 4ème ordre à l'aide des outils que nous avons développés dans les chapitres précédents. Nous allons comparer les algorithmes `RK45` et `mstz` sur cet exemple. Cette équation s'écrit :

$$\begin{cases} \dfrac{\partial u(t,x)}{\partial t} &= -\dfrac{\partial^2 u(t,x)}{\partial x^2} - \dfrac{\partial^4 u(t,x)}{\partial x^4} - \dfrac{1}{2}\dfrac{\partial u^2(t,x)}{\partial x} \quad 0 \leq t \leq 120 \quad x \in \mathbb{R} \\ u(0,x) &= u_0(x) \\ u(t,x) &= u(t,x+l) \end{cases} \tag{7.1}$$

C'est l'équation de Kuramoto-Sivashinsky (KS) à une dimension avec conditions aux limites périodiques. Cette équation apparaît dans de nombreux phénomènes comme les ondes de concentration [KT76], la propagation des flammes [Mic96] ou encore les turbulences hydrodynamiques [PZ85]. Elle a un comportement chaotique.

Le premier terme du membre de droite est comme celui de l'équation de la chaleur avec le signe opposé. Il amplifie les perturbations, qui sont stabilisées lorsqu'elles sont assez importantes par le terme d'ordre 4. Ce terme ressemble à celui de l'équation d'une poutre encastrée. Nous avons déjà rencontré le dernier terme dans l'équation de Burgers. C'est un terme de transport non-linéaire. Cette équation est décrite comme un bon problème test par Wanner et Hairer dans [HW96]. C'est un problème raide.

Jusqu'à maintenant, nous avons toujours utilisé des différences finies pour semi-discrétiser nos EDP. Dans le cas de l'équation KS, cette approche devient lourde à cause du terme de 4ème ordre. C'est pourquoi nous allons semi-discrétiser le problème à l'aide de la méthode dite *pseudo-spectrale* qui fait appel à la théorie de Fourier. Cette méthode permet aussi un traitement très aisé des conditions aux limites périodiques.

7.1 Quelques mots sur la théorie de Fourier

Dans cette section, nous introduirons succinctement les éléments de la théorie de Fourier qui nous seront utiles dans notre résolution de l'équation KS. Tous les détails peuvent être trouvés dans un livre standard sur le sujet, par exemple [GW99].

Soit $l > 0$ un réel.

Definition 7.1.1

$$L^2_P(0,l) = \{f : \mathbb{R} \to \mathbb{C},\ \textit{de période } l \textit{ et t.q. } \int_0^l |f(x)|^2 dx < \infty\}$$

L'indice P est là pour rappeler que les fonctions sont périodiques. Cet espace est muni du produit scalaire suivant :

$$(f,g) = \int_0^l f(x)\bar{g}(x)dx$$

Où $\bar{g}$ indique le conjugué complexe de g. La norme associée est :

$$\|f\|_2 = \sqrt{\int_0^l |f(x)|^2 dx}$$

L'idée de la théorie de Fourier est d'écrire une fonction comme une somme infinie de fonctions trigonométriques.

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n \mathrm{e}^{2i\pi nx/l} \text{ avec} \tag{7.2}$$

$$c_n = \frac{1}{l}\int_0^l f(x)\mathrm{e}^{-2i\pi nx/l}dx \tag{7.3}$$

Le membre de droite de l'équation (7.2) est appelé série de Fourier de la fonction f. Les c_n sont appelés coefficients de Fourier de la fonction périodique f. Pour effectuer des calculs numériques on doit se limiter à une somme finie. On définit :

$$f_N(x) = \sum_{n=-N}^{N} c_n \mathrm{e}^{2i\pi nx/l}$$

On a le théorème suivant :

Théorème 7.1.2 *Si $f \in L^2_P(0,l)$, l'approximation f_N définie ci-dessus tend vers f dans $L^2_P(0,l)$ quand $N \to \infty$. Autrement dit :*

$$\int_0^l |f(x) - f_N(x)|^2 dx \to 0 \textit{ quand } N \to \infty$$

Ceci implique notamment que $c_n \to 0$ quand $n \to \infty$.

Dans les calculs numériques, une fonction est presque toujours réduite à un certain nombre, fini, de valeurs ponctuelles. Aussi est-il important d'examiner si la convergence dans $\mathrm{L}^2_P(0,l)$ peut exprimer une égalité à x fixé.

Théorème 7.1.3 (Dirichlet) *Soit $f \in L^2_P(0,l)$, continue. Soit $x \in (0,l)$. Alors $f_N(x) \to f(x)$ quand $N \to \infty$.*

Le même théorème existe avec des hypothèses beaucoup plus faibles, mais les résultats sont alors plus complexes. Par exemple, supposons que la fonction f ait une discontinuité au point $x_0 \in (0, l)$. Soient

$$\begin{aligned} f^+(x_0) &= \lim\{f(x_0 + h),\ h > 0,\ h \to 0\} \\ f^-(x_0) &= \lim\{f(x_0 - h),\ h > 0,\ h \to 0\} \end{aligned}$$

Alors $f_N(x_0) \to \frac{1}{2}\big(f^+(x_0) + f^-(x_0)\big)$ quand $N \to \infty$.

Noter que les fonctions doivent être périodiques. En particulier, si l'on prend la série de Fourier d'une fonction f définie sur un intervalle borné et qu'ensuite l'on travaille sur ses coefficients de Fourier, il faut toujours garder à l'esprit que c'est comme si l'on travaillait sur la fonction f mais étendue par périodicité sur toute la droite réelle.

Voici deux propriétés importantes de la transformée de Fourier.

Propriété 7.1.4 *Si f est une fonction réelle alors $c_{-n} = \overline{c_n}$.*

Propriété 7.1.5 (Unicité) *Soient f et g, deux fonctions appartenant à $L^2_P(0, l)$. Soient c_n les coefficients de Fourier de f et d_n ceux de g. On a l'équivalence suivante :*

$$f = g \text{ presque partout } \Leftrightarrow c_n = d_n\ \forall n \in \mathbb{Z}$$

Dans les calculs numériques, on ne calcule pas l'intégrale (7.3) exactement. Par exemple, on peut employer la formule des rectangles (cf figure 7.1). Soit $0 = x_0 < x_1 < \ldots < x_M = l$ un maillage uniforme. Le pas de discrétisation h vaut l/M. Autrement dit, on a $x_j = \frac{j}{M} l \quad \forall j = 0, \ldots, M$. On note c_n^h l'approximation de c_n.

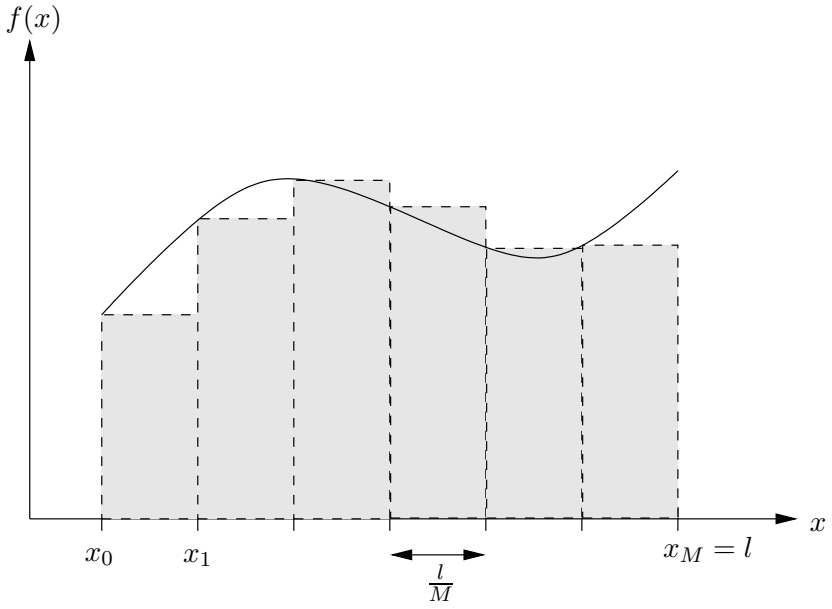


FIG. 7.1 – La formule du rectangle

$$c_n^h = \frac{1}{l}\sum_{j=0}^{M-1} f(x_j)\mathrm{e}^{-2i\pi n\frac{x_j}{l}}\frac{l}{M} \tag{7.4}$$

$$= \frac{1}{M}\sum_{j=0}^{M-1} f(x_j)\mathrm{e}^{-2i\pi n\frac{jl}{Ml}} \tag{7.5}$$

$$= \frac{1}{M}\sum_{j=0}^{M-1} f(x_j)\mathrm{e}^{-2i\pi n\frac{j}{M}} \tag{7.6}$$

Notons que :

$$\begin{aligned} c_{n+M}^h(t) &= \frac{1}{M}\sum_{j=0}^{M-1} f(x_j)\mathrm{e}^{-2i\pi(n+M)\frac{j}{M}} \\ &= \frac{1}{M}\sum_{j=0}^{M-1} f(x_j)\mathrm{e}^{-2i\pi n\frac{j}{M}}\mathrm{e}^{-2i\pi(M)\frac{j}{M}} \\ &= \frac{1}{M}\sum_{j=0}^{M-1} f(x_j)\mathrm{e}^{-2i\pi n\frac{j}{M}}\underbrace{\mathrm{e}^{-2i\pi j}}_{1} \\ &= c_n^h \end{aligned}$$

En particulier $c_{-n}^h = c_{M-n}^h$. Donc si notre maillage uniforme contient M subdivisions, alors il est inutile de calculer plus de M coefficients de Fourier. Le passage des M valeurs nodales $f(x_0), \ldots, f(x_{M-1})$ aux M coefficients de Fourier c_n est appelé transformée de Fourier discrète. Pour avoir autant de coefficients d'indices positifs que d'indices négatifs, nous prendrons toujours M impair. De plus, nous noterons $m = (M-1)/2$. On obtient alors une approximation de $f(x)$ de la manière suivante :

$$f(x) = \sum_{n=-m}^{m} c_n^h \mathrm{e}^{2i\pi nx/l}$$

En particulier

$$f(x_j) = \sum_{n=-m}^{m} c_n^h \mathrm{e}^{2i\pi n\frac{j}{M}}$$

Remarquer la similarité avec la formule (7.6). La transformation ci-dessus est appelée transformation de Fourier discrète inverse. Noter que $f(x_0) = f(x_M)$, ce qu'il nous fallait car la fonction f est périodique.

Pour calculer les M coefficients $c_n^h,\ n = -m, \ldots, m$ de la transformée de Fourier discrète, le nombre d'opérations semble être de l'ordre de M^2. Néanmoins, en 1965, Cooley et Tuckey (cf [CT65]) ont mis au point un algorithme qui permet d'effectuer ces calculs avec un nombre d'opérations de l'ordre de $M\log(M)$. Cet algorithme est connu sous le nom de F.F.T (pour Fast Fourier Transform). C'est lui qui permet vraiment d'employer la théorie de Fourier en analyse numérique. En effet, sans F.F.T., le temps de calcul est beaucoup trop long. On trouve une implémentation très efficace de cet algorithme dans MATLAB®, que nous noterons `fft`. Si l'on note $\mathbf{c^h} = (c_{-m}^h, \ldots, c_0^h, \ldots, c_m^h)$ et $\mathbf{f} = (f(x_0), \ldots, f(x_{M-1}))$ alors on a $\mathbf{c^h} = \texttt{fft}(\mathbf{f})$ (en fait, il y a encore une permutation des

indices et une multiplication par une constante, mais nous n'entrerons pas dans les détails ici, voir l'annexe A). De même, l'algorithme pour calculer la transformation de Fourier discrète inverse est noté `ifft`.

On peut remarquer que la formule de la transformée de Fourier discrète (7.6) peut être écrite matriciellement.

Definition 7.1.6 *Soit F, la matrice $M \times M$ t.q. $\forall \mathbf{f}$ $\texttt{fft}(\mathbf{f}) = F\mathbf{f}$. On a alors $\forall \mathbf{c}$ $\texttt{ifft}(\mathbf{c}) = F^{-1}\mathbf{c}$.*

Nous nous intéressons maintenant à $f'(x)$. Quels sont ses coefficients de Fourier ?

Théorème 7.1.7 *Si la fonction f, de période l, est continue sur $\mathbb{R}$ et si elle admet sur $[0, l]$ une dérivée, sauf éventuellement en un nombre fini de points et si de plus f' est continue par morceaux sur $[0, l]$ alors la série de Fourier de f' s'obtient en dérivant terme à terme celle de f :*

$$f'(x) = \sum_{n=-\infty}^{\infty} \frac{2i\pi n}{l} c_n e^{2i\pi nx/l}$$

7.2 La méthode pseudo-spectrale

Après les différences finies, la théorie de Fourier offre une autre façon de semi-discrétiser une équation aux dérivées partielles. Nous appliquerons la transformée de Fourier discrète à la variable spatiale x et cela nous donnera un schéma semi-discrétisé que nous pourrons résoudre à l'aide de l'algorithme `mstz`. La variable x appartient à l'intervalle $[0, l]$. Comme nous appliquons la théorie de Fourier seulement en espace, les coefficients de Fourier vont maintenant dépendre de t :

$$u(t,x) = \sum_{n=-\infty}^{+\infty} c_n(t) \mathrm{e}^{2i\pi n \frac{x}{l}} \tag{7.7}$$

$$\text{avec } c_n(t) = \frac{1}{l} \int_0^l u(t,x) \mathrm{e}^{-2i\pi n \frac{x}{l}} dx \tag{7.8}$$

Maintenant, comme vu plus haut, nous appliquons un maillage uniforme $0 = x_0 < \ldots < x_M = l$ sur $[0\ l]$. On obtient l'approximation :

$$u(t,x) \approx \sum_{n=-m}^{m} c_n^h(t) \mathrm{e}^{2i\pi n \frac{x}{l}} \tag{7.9}$$

$$\tag{7.10}$$

Nous appliquons maintenant la théorie de Fourier à l'équation KS. Pour alléger la notation, nous notons $q = \frac{2\pi}{l}$. En utilisant plusieurs fois le théorème 7.1.7 on obtient :

$$-\frac{\partial^2 u(t,x)}{\partial x^2} \approx \sum_{n=-m}^{m} n^2 q^2 c_n^h(t) \mathrm{e}^{inqx} \tag{7.11}$$

$$-\frac{\partial^4 u(t,x)}{\partial x^4} \approx \sum_{n=-m}^{m} (-n^4 q^4) c_n^h(t) \mathrm{e}^{inqx} \tag{7.12}$$

$$\frac{\partial u(t,x)}{\partial t} \approx \sum_{n=-m}^{m} \dot{c}_n^h(t) \mathrm{e}^{inqx} \tag{7.13}$$

Pour le terme de Burgers, on agit comme suit :

$$\text{On note } u^2(t,x) \approx \sum_{n=-m}^{m} d_n^h(t)\mathrm{e}^{inqx} \tag{7.14}$$

$$\text{alors } -\frac{1}{2}\frac{\partial u^2(t,x)}{\partial x} \approx -\frac{1}{2}\sum_{n=-m}^{m} inqd_n^h(t)\mathrm{e}^{inqx} \tag{7.15}$$

La relation entre $c_n^h(t)$ et $d_n^h(t)$ est faite comme suit. $\mathtt{ifft}(\mathbf{c^h}(t))$ donne les valeurs de la fonction $u(t,x)$ aux points x_j du maillage. Donc si l'on élève chaque composante de $\mathtt{ifft}(\mathbf{c^h}(t))$ au carré, on obtient les valeurs de $u^2(t,x)$ aux points x_j du maillage. Il suffit alors de prendre la transformée de Fourier discrète de ces valeurs pour obtenir les $d_n^h(t)$. Autrement dit :

$$\mathbf{d^h}(t) = \mathtt{fft}\Big(\mathtt{ifft}(\mathbf{c^h}(t)) : \mathtt{ifft}(\mathbf{c^h}(t))\Big)$$

où : dénote la multiplication composantes par composantes ($(-2\ 3\ 5) : (1\ 4\ 2) = (-2\ 12\ 10)$). En utilisant les équations (7.11), (7.12), (7.13) et (7.15) afin d'approximer l'équation KS (7.1), on obtient :

$$\sum_{n=-m}^{m} \dot{c}_n^h(t)\mathrm{e}^{inqx} = \sum_{n=-m}^{m} \left((n^2q^2 - n^4q^4)c_n^h(t) - \frac{1}{2}inqd_n^h(t)\right)\mathrm{e}^{inqx}$$

En utilisant la propriété 7.1.5 d'unicité des coefficients de Fourier, on obtient notre système semi-discrétisé :

$$\dot{c}_n^h(t) = (n^2q^2 - n^4q^4)c_n^h(t) - \frac{1}{2}inqd_n^h(t) \quad \forall n = -m, \ldots, m \tag{7.16}$$

Cette écriture permet mieux d'interpréter les actions des différents opérateurs. Si $n^2q^2 - n^4q^4 > 0$, i.e $n < 1/q$ alors les perturbations sont amplifiées. Par contre, pour les perturbations de hautes fréquence, $n^2q^2 - n^4q^4 < 0$ et alors les perturbations sont atténuées. Le dernier terme transporte les fréquences vers des fréquences plus hautes. Tout ce mécanisme assure que la solution est bornée et analytique sous certaines conditions (cf [CEES93]).

On peut écrire le système (7.16) sous forme vectorielle :

$$\dot{\mathbf{c}}^{\mathbf{h}}(t) = A\mathbf{c}^{\mathbf{h}}(t) - \frac{1}{2}inqF(F^{-1}\mathbf{c}^{\mathbf{h}}(t) : F^{-1}\mathbf{c}^{\mathbf{h}}(t))$$

avec $\mathbf{c}^{\mathbf{h}}(t) = (c_{-m}^h(t), \ldots, c_0^h(t), \ldots, c_m^h(t))$,

$$A = \begin{pmatrix} (-mq)^2 - (-mq)^4 & 0 & . & 0 \\ 0 & . & . & . \\ . & . & . & . \\ 0 & . & 0 & (mq)^2 - (mq)^4 \end{pmatrix}$$

et la matrice F est celle introduite en 7.1.6. Afin de pouvoir résoudre le problème dual, nous devons maintenant calculer le jacobien de ce système par rapport à $\mathbf{c}^{\mathbf{h}}(t)$. Ceci est quelque peu fastidieux (cf annexe A) mais le résultat est le suivant. On définit $a(t,\mathbf{c}) = A\mathbf{c} + F(F^{-1}\mathbf{c} : F^{-1}\mathbf{c})$. Voici le

jacobien de a par rapport à la deuxième variable dans le cas $m = 2$.

$$\nabla a(t, \mathbf{c}) = A - inq \begin{pmatrix} c_{-2} & c_2 & c_1 & c_0 & c_{-1} \\ c_{-1} & c_{-2} & c_2 & c_1 & c_0 \\ c_0 & c_{-1} & c_{-2} & c_2 & c_1 \\ c_1 & c_0 & c_{-1} & c_{-2} & c_2 \\ c_2 & c_1 & c_0 & c_{-1} & c_{-2} \end{pmatrix} \tag{7.17}$$

La généralisation pour tous les m se fait de la manière triviale. La matrice de droite est une matrice dite circulante (les diagonales sont constantes). Elle a certaines propriétés qui permettent d'accélérer quelque peu les calculs. Par exemple faire une transformation de Fourier est équivalent à diagonaliser une matrice circulante (cf [Ald01] et [Dav79] pour plus d'informations sur les matrices circulantes). Néanmoins, comme nous ne prendrons pas des valeurs de m très grandes, nous n'emploierons pas ces propriétés.

Nous sommes maintenant prêts à résoudre l'équation de KS. Mais tout d'abord afin de valider le code spectral, nous allons le tester sur l'équation de la chaleur et l'équation de Burgers.

7.3 Test du code spectral

7.3.1 La fonction g

Pour la fonction réelle g (rappel : on minimise $g(X(T)) - g(\bar{X}(T))$), nous faisons le raisonnement suivant. A partir des coefficients c_n^h, nous faisons la transformée de Fourier inverse, puis nous appliquons la même fonction g qu'à l'équation (6.7), à savoir une somme autour d'un point de contrôle $\bar{x}$ pondérée par une distribution normale. On a $w_j = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{\frac{1}{2}\left(\frac{x_j - \bar{x}}{\sigma}\right)^2}$ avec $\sigma = l/20$.

$$\begin{aligned} g(\mathbf{c^h}(T)) &= \sum_{j=0}^{M-1} \underbrace{\sum_{n=-m}^{m} c_n^h(T) \mathrm{e}^{iqnx_j}}_{u(T,x_j)} w_j = \sum_{n=-m}^{m} \left(\sum_{j=0}^{M-1} \mathrm{e}^{iqnx_j} w_j \right) c_n^h(T) \\ \nabla g(\mathbf{c^h}(T))_n &= \sum_{j=0}^{M-1} \mathrm{e}^{iqnx_j} w_j = \sum_{j=0}^{M-1} \mathrm{e}^{2i\pi n \frac{j}{M}} w_j \end{aligned}$$

Considérons les w_j comme des coefficients de Fourier. On définit donc (pour $n = 0, \ldots, m$) $w_{-n} = w_{M-n}$. On définit maintenant $\mathbf{w} = (w_{-m}, \ldots, w_m)$. On a alors :

$$\nabla g(\mathbf{c^h}(T)) = \texttt{ifft}(\mathbf{w})$$

7.3.2 Visualisation du dual

Nous aimerions visualiser le dual dans l'espace des x, comme nous le faisions auparavant, afin de voir quelles portions de $[0, l]$ sont importantes au cours du temps pour obtenir la solution. Néanmoins, maitenant le dual est dans l'espace de Fourier. Pour passer des coefficients de Fourier $c_{-m}^h(t), \ldots, c_m^h(t)$ à la fonction de départ $u(t, x)$, on fait simplement une transformée de Fourier

inverse. On fait de même pour l'estimation de l'erreur qui est dans le même espace que les $c_n^h(t)$. Peut-on faire de même pour le dual ? On a :

$$\begin{aligned} g(\mathbf{c^h}(T)) - g(\bar{\mathbf{c}}^{\mathbf{h}}(T)) &\approx \sum_{\text{pas de temps}} \left(\bar{e}(t_n), \bar{\psi}(t_n)\right) \\ &= \sum_{\text{pas de temps}} \left(F^{-1}\bar{e}(t_n), F^T\bar{\psi}(t_n)\right) \end{aligned}$$

$F^{-1}\bar{e}(t_n)$ dénote la transformée de Fourier inverse de $\bar{e}$, i.e. sa forme dans l'espace de départ. Donc pour voir comment le dual est actif dans l'espace de départ il faut lui appliquer l'opération F^T.

Remarque 7.3.1 *Attention, dans les lignes qui précèdent, $\bar{e}$ indique une approximation de e et non son conjugué complexe.*

7.3.3 L'équation de la chaleur

La semi-discrétisation spectrale de l'équation de la chaleur donne le système suivant :

$$\dot{c}_n^h(t) = -\nu n^2 q^2 c_n^h(t)$$

On remarque que tous les modes sont indépendants. Il est facile de calculer la solution exacte de ce problème semi-discrétisé.

$$c_n^h(t) = c_n^h(0)\mathrm{e}^{-\nu n^2 q^2 t}$$

Nous prenons les mêmes paramètres qu'à la section 6.2, i.e. $h = 1/15$, $\Delta t = 0.01$, $\nu = 0.1$, $\bar{x} = 0.4$ et $u(0,x) = \sin(2\pi x)$. L'avancement en temps se fait avec la méthode d'Euler progressive. L'avantage de cette condition initiale est qu'on peut l'écrire :

$$\sin(2\pi x) = -\frac{1}{2}i\mathrm{e}^{2i\pi x} + \frac{1}{2}i\mathrm{e}^{-2i\pi x}$$

Et donc $c_1(0) = -0.5i$, $c_{-1}(0) = 0.5i$ et tous les autres coefficients de Fourier sont nuls. Donc le fait de ne considérer que M coefficients de Fourier au lieu de tous n'est pas une approximation. On le constate dans le tableau 7.1 des résultats, où l'erreur en espace $\mathcal{E}^h$ est quasi nulle. La solution exacte a été calculée en prenant la transformée de Fourier de la solution exacte (6.5).

TOL	$\mathcal{E}$	$\mathcal{E}^h$	$\mathcal{E}^t$	$\overline{\mathcal{E}}^t$	# eval. a	# eval. $(a')^*$	CPU	N	it
0.001	0.000793	-3.89e-16	0.000793	0.000792	4500	1500	2.47	800	4

TAB. 7.1 – Résultats pour la semi-discrétisation spectrale de l'équation de la chaleur

On remarque que les résultats sont très similaires en terme de temps de calcul à ceux obtenus avec une semi-discrétisation par différences finies du tableau 6.1. Le point crucial est qu'ici, la semi-discrétisation est exacte. Néanmoins, si l'on prend une condition initiale moins régulière, alors la semi-discrétisation par la méthode spectrale ne sera plus exacte, car on n'aura plus la propriété de n'avoir que 2 coefficients non nuls.

Un dernier point qu'il faut avoir à l'esprit est qu'en employant la méthode spectrale, on ne traite pas des mêmes conditions aux bords. Par la semi-discrétisation par différences finies, on traitait la condition $u(t,0) = u(t,1) = 0 \ \ \forall t$. Ici, l'on traite simplement $u(t,0) = u(t,1) \ \ \forall t$, la condition aux bords périodique. Ces deux conditions sont équivalentes pour le problème de la chaleur avec la condition initiale que nous avons pris ci-dessus. Néanmoins, en général ces deux conditions sont différentes et on l'observera par la suite. On peut d'ailleurs observer ceci en comparant les figures 7.2 et 6.1. Le dual au temps 0 à une forme différente. Pour la semi-discrétisation spectrale, il est constant et égal à 1. Pour les différences finies, c'est une parabole qui vaut 0 pour $x = 0$ et $x = 1$ et a un maximum en $x = 0.5$. Ceci est dû aux conditions aux bords différentes.

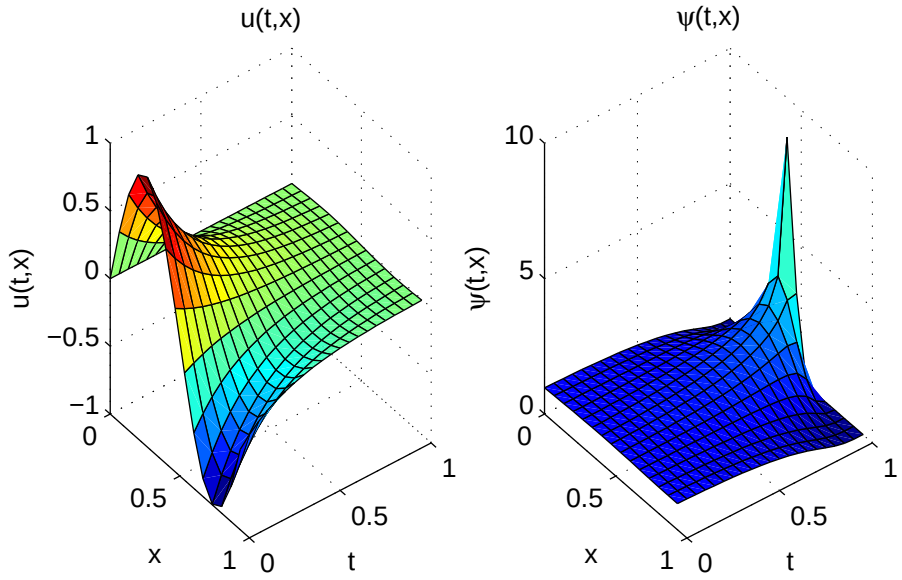


FIG. 7.2 – Les résultats pour l'équation de la chaleur et son dual, semi-discrétisation spectrale

7.3.4 L'équation de Burgers

Nous appliquons maintenant une semi-discrétisation spectrale à l'équation de Burgers visqueuse (6.9). En suivant le même raisonnement que pour l'équation KS (7.16), le système semi-discrétisé auquel on aboutit est

$$\dot{c}_n^h(t) = (-\varepsilon n^2 q^2) c_n^h(t) - \frac{1}{2} inq d_n^h(t) \ \ \forall n = -m, \ldots, m$$

où $d_n^h(t)$ sont les coefficients de Fourier de $u^2(t,x)$. Nous prenons les mêmes conditions initiales et les mêmes paramètres qu'à la section 6.4, c'est-à-dire $\varepsilon = 0.01$, $h = 0.025$, $\Delta t = 0.005$ et le point de contrôle $\bar{x} = 0.5$. L'avancement en temps se fait avec la méthode d'Euler progressive. On obtient les figures 7.4 et 7.5.

Les résultats sont assez similaires à ceux trouvés par la méthode des différences finies. Le dual a la même forme. On remarque que cette fois les conditions aux bords périodiques prennent toute leur importance. En effet, comme souligné en début de chapitre, la méthode de Fourier étend par périodicité une fonction définie sur un intervalle borné à toute la droite réelle. Donc, dans le cas de l'escalier descendant, la fonction que "voit" la semi-discrétisation spectrale est celle de la figure 7.3.

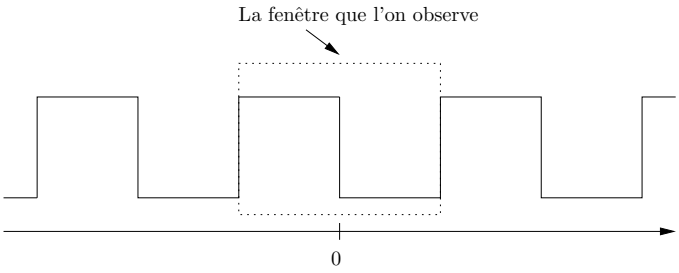
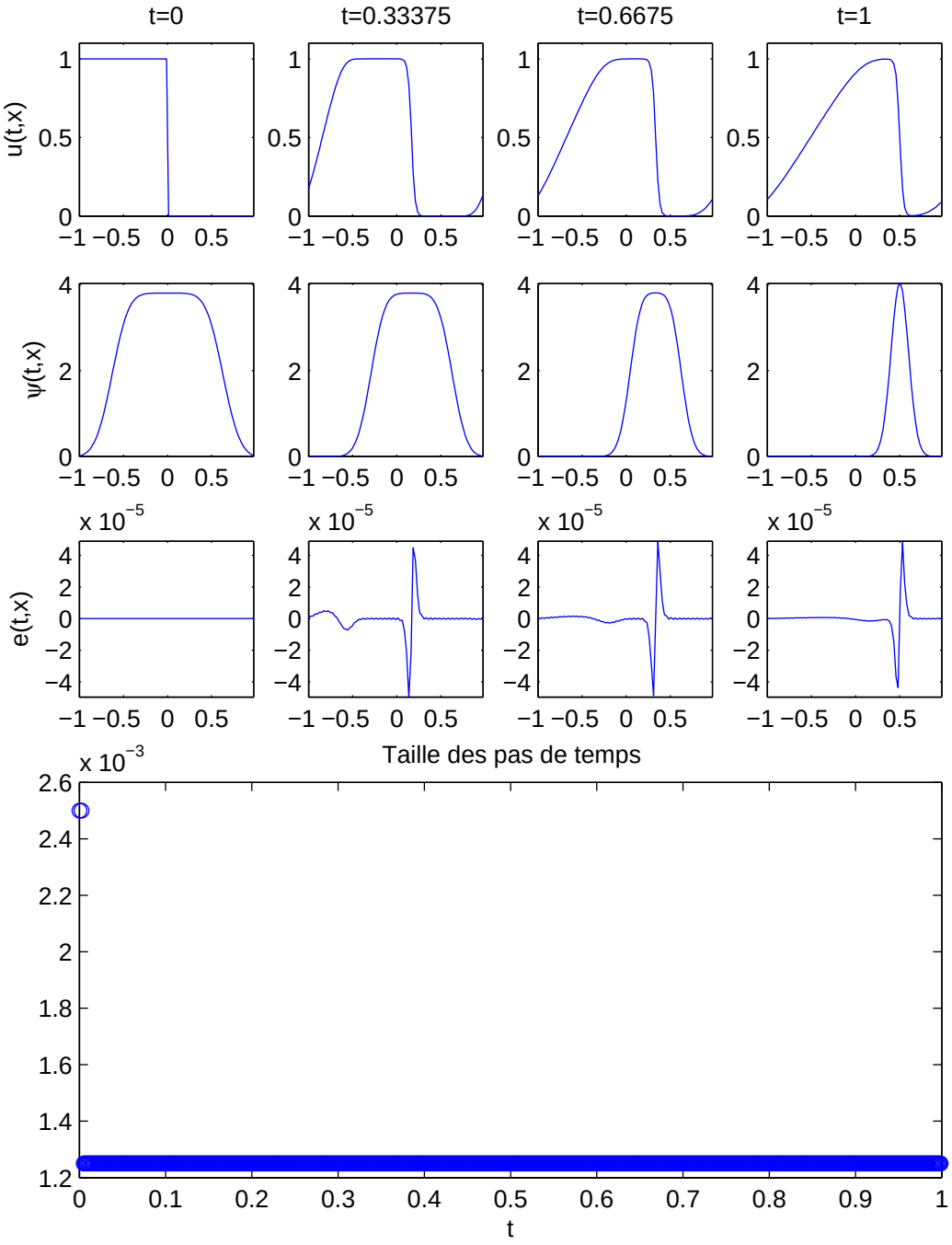


FIG. 7.3 – L'escalier descendant, étendu par périodicité à la droite réelle

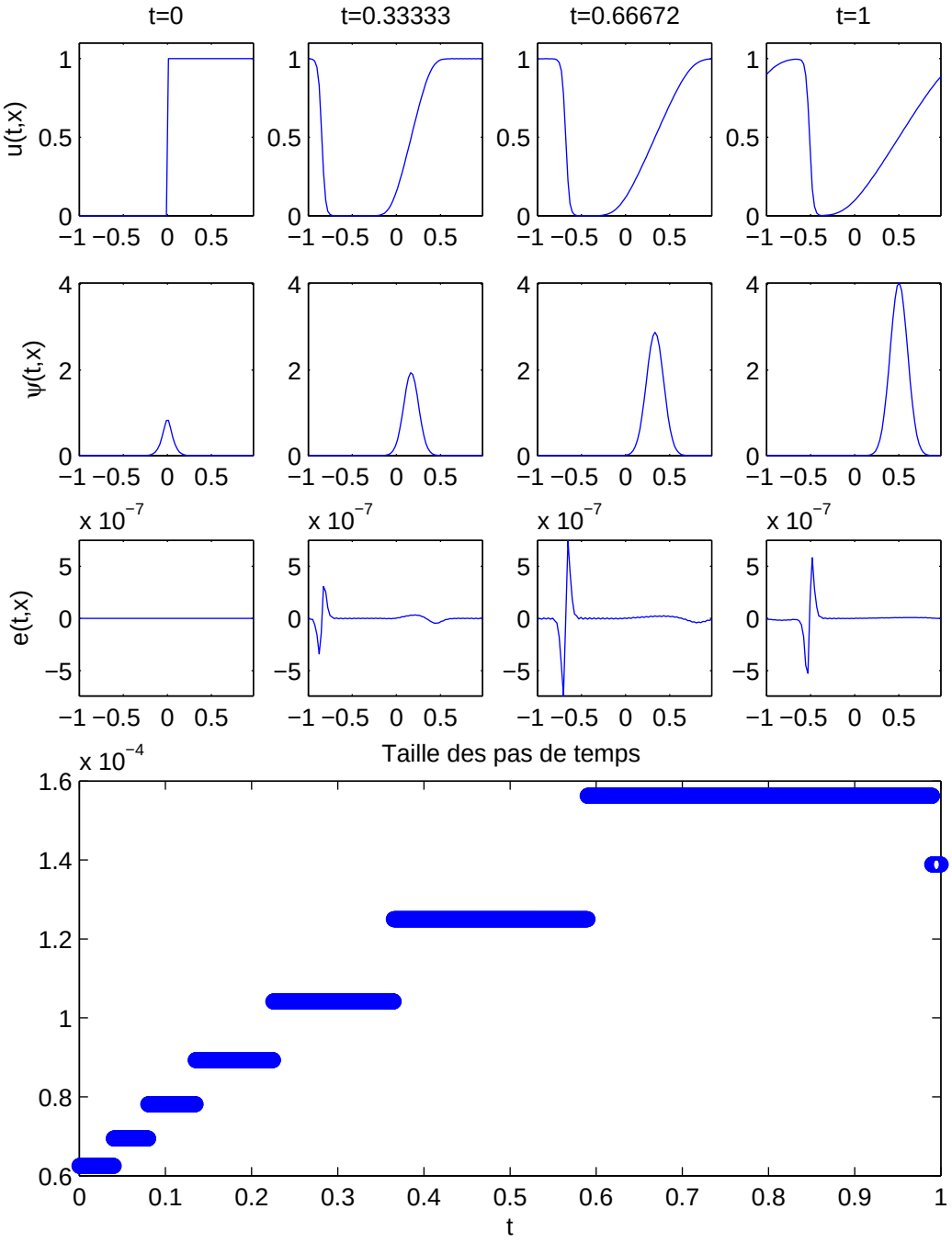
On a donc un escalier descendant, suivi d'un escalier montant et ainsi de suite jusqu'à l'infini. Cela explique les résultats qu'on voit dans les figures 7.4 et 7.5. On observe simultanément une onde de choc et une onde de raréfaction. Néanmoins, comme notre point de contrôle $\bar{x} = 0.5$ se trouve dans la zone affectée par l'onde de choc dans le cas de l'escalier montant et de l'onde de raréfaction dans le cas de l'escalier descendant, les résultats sont similaires à ceux observés dans le cas des différences finies. Le raffinement quelque peu différent de l'escalier descendant est dû aux interférences apportées par l'onde de raréfaction. En effet, celle-ci demandant beaucoup plus de calculs elle prend quelque peu le dessus sur l'onde de choc. Dans le cas de l'escalier montant, l'effet de l'onde de choc supplémentaire est insignifiant.

On remarque que le temps de calcul est plus long que pour la méthode des différences finies. Ceci est principalement dû aux calculs faits pour le dual. En effet, l'évaluation du dual est chère car il faut à chaque fois construire la matrice circulante vue en (7.17). Dans le cas des différences finies, la matrice qui intervient dans le calcul du dual n'a que 3 diagonales non nulles.



	TOL	$\overline{\mathcal{E}}^t$	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	0.005	-0.00397	4191	1397	10.97	799	3

FIG. 7.4 – Problème de Burgers dans le cas de l'onde de choc, semi-discrétisation spectrale



	TOL	$\overline{\mathcal{E}}^t$	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	0.005	0.00371	49551	16517	151.4	8705	5

FIG. 7.5 – Problème de Burgers dans le cas de l'onde de raréfaction, semi-discrétisation spectrale

7.4 Application à Kuramoto-Sivashinsky

Maintenant que nous avons vérifié que notre code spectral fonctionnait bien, nous allons l'appliquer à l'équation de Kuramoto-Sivashinsky. Un des problèmes d'une telle équation est qu'il est très difficile de savoir si le résultat que l'on obtient est stable ou si c'est juste une amplification des erreurs d'arrondis. Pour résoudre ce problème, il faut observer l'estimation d'erreur. Quand le résultat est non stable, elle est très grande bien que parfois visuellement, on ne s'aperçoive pas des instabilités.

Suivant [HW96], nous avons posé $q = 0.025$, ce qui donne que la période est $l = \frac{2\pi}{q} \approx 251$. Nous avons pris $h = l/151$, ce qui donne les coefficients de Fourier pour les valeurs de n de -75 à $+75$. Comme $n^2q^2 - n^4q^4$ vaut 0 pour $n = 40$, cela nous a paru suffisant. En effet, le terme d'ordre 4 amortit tous les coefficients de Fourier pour $|n| > 40$. Le temps final est $T = 120$ et le pas de temps initial est $T/400 = 0.3$. L'avancement en temps se fera avec la même méthode Runge-Kutta d'ordre 5 qu'au chapitre 4. La condition initiale est la même que celle de [HW96], c'est-à-dire

$$u(0,x) = 16\max(0,\eta_1,\eta_2,\eta_3,\eta_4), \quad \begin{array}{l} \eta_1 = \min(x/l, 0.1 - x/l) \\ \eta_2 = 20(x/l - 0.2)(0.3 - x/l) \\ \eta_3 = \min(x/l - 0.6, 0.7 - x/l) \\ \eta_4 = \min(x/l - 0.9, 1 - x/l) \end{array}$$

De plus, nous prenons la même fonction g qu'auparavant. Le point de contrôle est $\bar{x} = 70$.

Le schéma ne parvient pas à résoudre l'équation KS de manière stable. Il faudrait employer un nombre plus important de coefficients de Fourier. Néanmoins, ceci augmente grandement le temps de calcul. Nous adoptons une autre stratégie : nous modifions l'équation KS de la manière suivante.

$$\frac{\partial u(t,x)}{\partial t} = -\lambda\frac{\partial^2 u(t,x)}{\partial x^2} - \frac{\partial^4 u(t,x)}{\partial x^4} - \frac{1}{2}\frac{\partial u^2(t,x)}{\partial x}$$

Pour $\lambda = 1$, on obtient l'équation initiale (7.1). En diminuant, la valeur du paramètre λ, on diminue la manière dont le terme "inverse chaleur" amplifie les perturbations. Ainsi, on devrait pouvoir résoudre l'équation sans devoir employer plus de coefficients de Fourier. Dans le tableau 7.2, on peut voir les résultats obtenus pour les valeurs de λ entre 1 et 0.9.

λ	TOL	$\overline{\mathcal{E}}^t$	CPU	N	it
1	0.1	-	158.75	400	1
0.99	0.1	-	160.54	400	1
0.98	0.1	-	128.88	400	1
0.97	0.1	-	142.49	400	1
0.96	0.1	-	136.72	400	1
0.95	0.1	0.0062	101.98	508	2
0.94	0.1	-	112.68	400	1
0.93	0.1	-0.0239	44.88	400	1
0.92	0.1	-3.81e-05	44.86	400	1
0.91	0.1	0.000396	45.03	400	1
0.9	0.1	-6.1e-05	44.82	400	1

TAB. 7.2 – Effet des différentes valeurs de λ. "-" signifie que l'estimation de l'erreur est infinie

On remarque que l'on obtient de bonnes valeurs pour $\lambda \leq 0.93$ et $\lambda = 0.95$. Nous décidons donc de prendre $\lambda = 0.93$ pour la suite. Maintenant que nous avons établi quelles valeurs nous allions donner aux paramètres, nous faisons un calcul très précis avec une tolérance de 10^{-11} pour avoir une valeur de référence. C'est par rapport à cette valeur que seront calculées les vraies erreurs $\mathcal{E}^t$ des tableaux de résultats.

Nous comparons maintenant les algorithmes `mstz` et `RK45` sur ce problème test. On obtient les résultats du tableau 7.3.

	TOL	$\mathcal{E}^t$	$\overline{\mathcal{E}}^t$	# eval. a	# eval. $(a')^*$	CPU	N	it
`mstz`	0.0001	2.09e-05	3.41e-05	18666	6222	115.94	637	2
`RK45`	0.0001	3.55e-05	-	29499	0	55.49	2382	3
`mstz`	1e-05	-8.01e-06	-8.61e-06	21078	7026	133.53	771	2
`RK45`	1e-05	4.2e-06	-	45915	0	86.2	3726	3
`mstz`	1e-06	4.61e-08	-6.26e-07	25776	8592	158.04	1032	2
`RK45`	1e-06	8.22e-07	-	71853	0	138.16	5855	3
`mstz`	1e-07	4.15e-07	3.24e-09	79254	26418	484.8	2524	3
`RK45`	1e-07	-	-	-	-	-	-	-

TAB. 7.3 – Comparaison des algorithmes `mstz` et `RK45` sur l'équation de Kuramoto-Sivashinsky.

On remarque que l'algorithme `mstz` a besoin de moins de pas de temps et de moins d'évaluations de la fonction a pour arriver à une tolérance donnée. Néanmoins, il a besoin d'un temps de calcul plus grand car les évaluations de la fonction $(a')^*$ (cf (7.17)) sont très chères. Mais n'oublions pas que grâce à cela, nous avons une estimation de l'erreur, absente dans l'algorithme `RK45`. De plus, pour la tolérance 10^{-7}, l'algorithme `RK45` n'arrive pas à résoudre l'équation de Kuramoto-Sivashinsky. En effet, la tolérance qu'il demande sur l'erreur locale doit être plus petite que 10^{-15}, ce qu'il n'arrivera jamais à obtenir car il faut demander des pas de temps trop petits. Une fois encore, on voit que l'algorithme `RK45` avance aveuglément en temps. Il borne seulement l'erreur locale et quand il remarque qu'il n'arrivera pas sous la tolérance TOL, il veut raffiner. Mais c'est trop tard, le raffinement nécessaire est trop petit. Il fallait raffiner avant. Mais pour cela il faut avoir une vision globale de l'intervalle temporel, ce qu'offre l'algorithme `mstz` et son dual.

On remarque que pour la tolérance de 10^{-7}, l'estimation de l'erreur est beaucoup trop petite. En effet, on se rapproche de la tolérance 10^{-11} avec laquelle a été calculée la solution exacte. Il se peut aussi que ce soit le fait d'employer des demi-pas qui produise ce problème. Avec la version basique, on obtient une estimation d'erreur de 1.19e-08 pour une vraie erreur de 4.25e-07. C'est meilleur mais toujours trop petit.

Pour terminer, on peut observer dans la figure 7.7, la solution de l'équation de Kuramoto-Sivashinsky que nous avons trouvée pour une tolérance TOL= 10^{-6}. C'est une belle illustration du chaos. Néanmoins, on sait, grâce à l'estimation de l'erreur a posteriori, que c'est la vraie solution du système semi-discrétisé. Cette belle assurance nous est donnée par l'algorithme `mstz` qui s'est révélé très fiable tout au long de notre travail.

Dans la figure 7.8, on peut observer le dual. On remarque qu'il est très grand au début et diminue rapidement. Ceci indique une grande dépendance aux conditions initiales, c'est-à-dire que le problème est chaotique. On peut aussi observer la taille des pas de temps dans la figure 7.6.

On remarque bien l'emploi de petits pas de temps au début, comportement similaire à celui observé sur le problème de Lorenz.

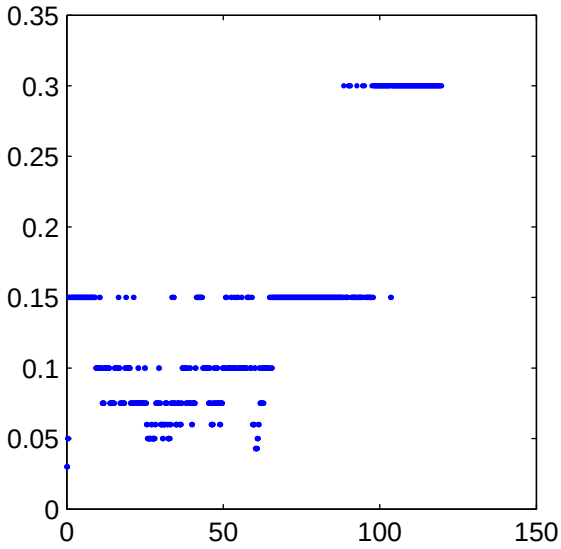


FIG. 7.6 – La taille des pas de temps

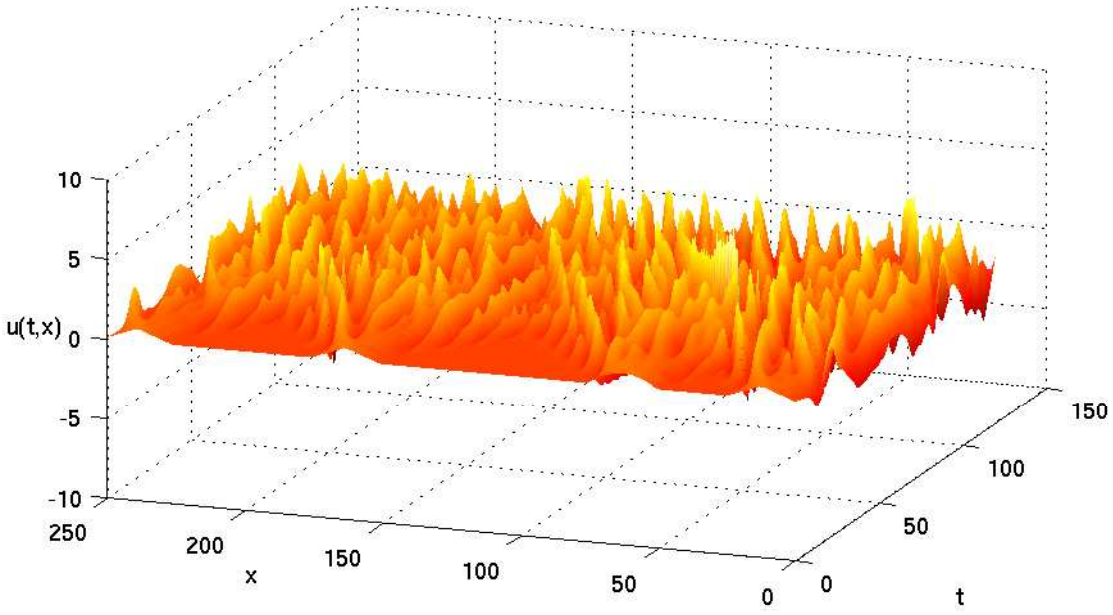


FIG. 7.7 – La solution de KS

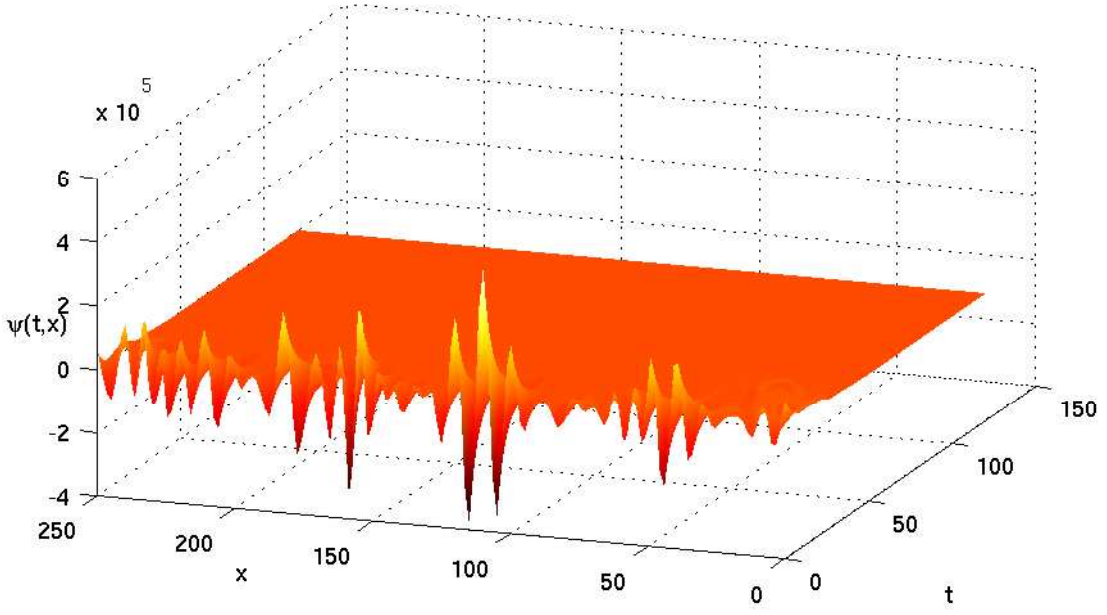


FIG. 7.8 – Le dual de KS

Chapitre 8

Conclusion

L'algorithme `mstz` a donné des résultats très positifs. Sur les problèmes chaotiques que nous avons testés, c'est-à-dire la transition à la turbulence, le problème de Lorenz et l'équation de Kuramoto-Sivashinsky, `mstz` emploie 2 à 3 fois moins d'évaluations de la fonction donnée a que l'algorithme `RK45` basé sur la méthode de Runge-Kutta adaptative développée par Dormand et Prince. De plus, l'algorithme `mstz` donne une estimation a posteriori de l'erreur globale qui permet de savoir si l'on peut faire confiance aux résultats obtenus. C'est une question essentielle pour les problèmes chaotiques. En particulier, on peut savoir si il est possible de résoudre tel ou tel problème numériquement. En effet, on peut mettre en évidence que des problèmes "trop" chaotiques sont impossibles à résoudre par `mstz` sur un ordinateur avec une précision donnée.

Par ailleurs, notre version de l'algorithme `mstz` emploie 2 à 3 fois moins de pas de temps au total que celle développée par Szepessy et al. En effet, elle a quelques propriétés supplémentaires : elle adapte le facteur de raffinement et elle résout le problème dual avec une précision moindre que le problème de départ.

L'application à l'équation de Burgers a révélé quelques comportements étonnants de cette équation, comme le fait qu'une onde de raréfaction est plus difficile à résoudre qu'une onde de choc. Cela a confirmé des résultats récents dans ce domaine.

Pour terminer, l'application à l'équation de Kuramoto-Sivashinsky, une EDP chaotique, a mis en évidence la qualité de `mstz`. En effet, pour cette équation, il est difficile de savoir si l'on peut faire confiance aux résultats fournis par une méthode numérique. `mstz` négocie bien ce type de problèmes et donne une solution à cette équation.

Comme points négatifs, on notera que `mstz` a besoin de plus de mémoire qu'une méthode de type "Runge-Kutta adaptatif". De plus, il faut le jacobien $(a')^*$ qui n'est pas toujours disponible et est parfois long à calculer, en particulier pour les EDP. Pour des problèmes non chaotiques, `mstz` est un peu moins efficace que les méthodes de type "Runge-Kutta adaptatif". En effet, les mécanismes mis en place pour traiter le chaos, prennent du temps de calcul inutile pour les cas non chaotiques. Néanmoins, cette différence (environ 20% d'évaluations de a supplémentaires) est compensée par la présence d'une estimation de l'erreur commise.

Annexe A

Le dual de Kuramoto-Sivashinsky

Nous allons maintenant calculer la fonction $(a')^*$ dans le cas de l'équation de Kuramoto-Sivashinsky. Dans ce chapitre, nous omettrons l'indice h dans $\mathbf{c^h}$ et noterons simplement $\mathbf{c}$. De plus, nous numéroterons les coefficients de Fourier d'une autre manière. Au lieu d'avoir

$$\mathbf{c} = (c_{-m}, \ldots, c_0, \ldots, c_m)$$

Nous prendrons

$$\mathbf{c} = (c_0, c_1, \ldots, c_m, c_{-m}, \ldots, c_{-1})$$

Donc depuis ici, quand nous parlerons de c_1, cela signifiera c_0, c_2 signifiera c_1 et ainsi de suite. Avec cet ordre des indices, la matrice F définie en 7.1.6 a maintenant l'agréable forme suivante :

$$F_{nj} = \frac{1}{M} \mathrm{e}^{-2i\pi(n-1)\frac{j-1}{M}}$$

En particulier, $F^{-1} = M\overline{F}$, la matrice est à une constante près orthogonale. On a :

$$\begin{aligned} \dot{\mathbf{c}}(t) &= a(t, \mathbf{c}(t)) \\ \text{avec } a(t, \mathbf{c}) &= A\mathbf{c} - \frac{1}{2} inqF(M\overline{F}\mathbf{c} : M\overline{F}\mathbf{c}) = A\mathbf{c} - \frac{1}{2} M^2 inqF(\overline{F}\mathbf{c} : \overline{F}\mathbf{c}) \end{aligned}$$

où la matrice A vaut maintenant :

$$A = \begin{pmatrix} 0 & 0 & 0 & . & . & . & . & . \\ 0 & q^2 - q^4 & 0 & . & . & . & . & . \\ 0 & 0 & (2q)^2 - (2q)^4 & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & (mq)^2 - (mq)^4 & 0 & . & . \\ . & . & . & . & 0 & (-mq)^2 - (-mq) & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & (-q)^2 - (-q)^4 \end{pmatrix}$$

On définit

$$g(\mathbf{c}) = F(\overline{F}\mathbf{c} : \overline{F}\mathbf{c})$$

Ainsi

$$g_n(\mathbf{c}) = \sum_{j=1}^{M} F_{nj}(\overline{F}\mathbf{c} : \overline{F}\mathbf{c})_j \tag{A.1}$$

$$= \sum_{j=1}^{M} F_{nj}\left(\sum_{k=1}^{M}\overline{F}_{jk}c_k\right)^2 \tag{A.2}$$

$$\left[\nabla g_n(\mathbf{c})\right]_s = 2\sum_{j=1}^{M} F_{nj}\left(\sum_{k=1}^{M}\overline{F}_{jk}c_k\right)\overline{F}_{js} \tag{A.3}$$

$$= 2\sum_{j=1}^{M}\sum_{k=1}^{M} F_{nj}\overline{F}_{js}\overline{F}_{jk}c_k \tag{A.4}$$

Soit

$$\mathbf{p} = \overline{F}\mathbf{c} \text{ ainsi } p_j = \sum_{k=1}^{M}\overline{F}_{jk}c_k$$

Ainsi, à partir de (A.4), on obtient

$$\left[\nabla g_n(\mathbf{c})\right]_s = 2\sum_{j=1}^{M} F_{nj}\overline{F}_{js}p_j \tag{A.5}$$

$$= \frac{2}{M^2}\sum_{j=1}^{M} \mathrm{e}^{-2i\pi(j-1)\frac{n-s}{M}}p_j \tag{A.6}$$

Maintenant, nous distinguons 2 cas. Premier cas : $n \geq s$, ainsi $0 \leq n-s \leq M-1$.

$$\left[\nabla g_n(\mathbf{c})\right]_s = \frac{2}{M}\sum_{j=1}^{M} F_{(n-s+1)j}p_j$$

Soit $\mathbf{z} = F\mathbf{p}$. Alors $\left[\nabla g_n(\mathbf{c})\right]_s = \frac{2}{M}z_{n-s+1}$ si $n \geq s$. Mais

$$\mathbf{z} = F\mathbf{p} = F\overline{F}\mathbf{c} = \frac{1}{M}\mathbf{c} \Rightarrow \left[\nabla g_n(\mathbf{c})\right]_s = \frac{2}{M^2}c_{n-s+1} \text{ si } n \geq s$$

Le deuxième cas est $n < s$, alors $-M+1 \leq n-s \leq -1$. De (A.6),

$$\left[\nabla g_n(\mathbf{c})\right]_s = \frac{2}{M^2}\sum_{j=1}^{M} \mathrm{e}^{2i\pi(j-1)\frac{s-n}{M}}p_j$$

$$= \frac{2}{M}\sum_{j=1}^{M}\overline{F}_{(-n+s+1)j}p_j$$

Soit $\mathbf{w} = \overline{F}\mathbf{p}$. Alors $\left[\nabla g_n(\mathbf{c})\right]_s = \frac{2}{M}w_{-n+s+1}$ si $n < s$. Mais $\mathbf{w}$ a la propriété suivante :

$$w_j = \frac{1}{M}c_{M+2-j} \text{ pour } j \geq 2$$

En effet :

$$\begin{aligned}
\frac{1}{M}c_{M+2-j} &= z_{M+2-j} \\
&= \sum_{k=1}^{M} F_{(M+2-j)k}p_k \\
&= \frac{1}{M}\sum_{k=1}^{M} \mathrm{e}^{-2i\pi(M+1-j)\frac{k-1}{M}}p_k \\
&= \frac{1}{M}\sum_{k=1}^{M} \mathrm{e}^{2i\pi(j-1)\frac{k-1}{M}}\underbrace{\mathrm{e}^{-2i\pi M\frac{k-1}{M}}}_{1}p_k \\
&= \sum_{k=1}^{M} \overline{F}_{jk}p_k = w_j
\end{aligned}$$

En résumé :

$$\begin{aligned}
\left[\nabla g_n(\mathbf{c})\right]_s &= \frac{2}{M^2}c_{n-s+1} \text{ si } n \geq s \\
\left[\nabla g_n(\mathbf{c})\right]_s &= \frac{2}{M^2}c_{M+n-s+1} \text{ si } n < s
\end{aligned}$$

Sous forme matricielle :

$$\nabla g(\mathbf{c}) = \frac{2}{M^2}\begin{pmatrix}
c_1 & c_M & c_{M-1} & . & c_2 \\
c_2 & c_1 & c_M & . & c_3 \\
c_3 & c_2 & c_1 & . & c_4 \\
. & . & . & . & . \\
c_M & c_{M-1} & c_{M-2} & . & c_1
\end{pmatrix}$$

On remarque que les diagonales sont constantes. On a donc que le jacobien de a par rapport à sa deuxième variable vaut :

$$\nabla a(t,\mathbf{c}) = A - \frac{1}{2}M^2 inq\nabla g(\mathbf{c}) = A - inq\begin{pmatrix}
c_1 & c_M & c_{M-1} & . & c_2 \\
c_2 & c_1 & c_M & . & c_3 \\
c_3 & c_2 & c_1 & . & c_4 \\
. & . & . & . & . \\
c_M & c_{M-1} & c_{M-2} & . & c_1
\end{pmatrix}$$

Annexe B

Code Matlab

```
0001 %% MSTZ
0002 function [T,X, varargout]=mstz(TOL, flux, Tin, Tfin, X0, g, N1, methodflag)
0003 % mstz(TOL, flux, Tin, Tfin, X0, g, N1, methodflag)
0004 % Solveur d'EDO basé sur la principe variationnel de Szepessy, Moon, Tempone et Zouraris.
0005 % pour resoudre X'(t)=flux(t,x) avec X(Tin)=X0 et t dans [Tin Tfin]
0006 %
0007 % TOL, la tolerance que l'on demande pour g(Xvrai(T))-g(X(T)) ou Xvrai est la vraie solution. Cette erreur
0008 % est estimée par l'algorithme.
0009 %
0010 % flux, le flux. Doit renvoyer un vecteur-colonne de la meme taille que X0. Il faut qu'il existe une fonction
0011 % flux_prime_star qui est la transposée de la dérivée de flux(t,x) par rapport à  x. C'est donc une matrice.
0012 %
0013 % Tin, Tfin, les temps initiaux et finaux
0014 %
0015 % X0 la condition au temps Tin. Vecteur colonne
0016 % g, une fonction qui prend en argument un vecteur de meme taille que X0 et renvoie un réel. On mininimise
0017 % g(Xvrai(T))-g(X(T)) ou Xvrai est la vraie solution.
0018 %
0019 % N1, le nombre de pas initiaux.
0020 %
0021 % methodflag. Pour choisir quelle version de l'algorithme on emploie :
0022 %     1    : version basique (M=10, floor)
0023 %     2    : version basique avec ceil
0024 %     3    : version avec M=2 toujours
0025 %     4    : version avec dt* sans raffinement compensé (M=5)
0026 %     5    : version avec dt* et raffinement compensé
0027 %     6    : version avec dt* et raffinement compensé et floor
0028 %     7    : version avec dt* sans raffinement compensé (M=5) et floor
0029 %     8    : avec deraffinement
0030 %     9    : avec deraffinement et floor
0031 %    10    : avec demi-pas. Autrement dit, on prend l'estimation avec le demi-pas comme vraie et on
0032 % estime l'erreur avec le pas normal. Le reste est basique
0033 %    11    : pour faire les figures.
0034 % T est les pas de temps finaux (vecteur colonne)
0035 % X est l'estimation de la solution. Chaque ligne correspond à  l'estimation de la solution pour le pas
0036 % de temps correspondant dans T.
0037 % Pour les paramètres retournés, il y plusieurs possibilités :
0038 % [T,X,psi]=mstz(...) donne T, X et la solution finale du dual.
0039 % [T,X,psi,e]=mstz(...) donne T, X, la solution du dual et l'estimation des erreurs locales
0040 % [T,X,e, est, evaltot, evalfin, evaltotd, evalfind, it] pour 9 arguments de sortie
0041 % [T,X,psi,e, est, evaltot, evalfin, evaltotd, evalfind, it] pour 10 arguments de sortie
0042 % est : estimation de l'erreur globale
0043 % evaltot : nombre d'évaluation de la fonction flux en tout
0044 % evalfin : nombre d'évaluation de la fonction flux lors de la dernière itération
0045 % evaltotd: nombre d'évaluation de la fonction flux_prime_star en tout
0046 % evalfind: nombre d'évaluation de la fonction flux_prime_star lors de la dernière itération
0047
0048 global YES NO;
```

```
0049 YES=1;
0050 NO=0;
0051
0052
0053 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054 % Affectation des paramètres        %
0055 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056
0057
0058 debug=NO; % pour les display, les plot, et tout et tout...
0059 flagflag; % pour avoir la correspondance entre les numéros ci-dessus pour methodflag et les noms
0060
0061 if(methodflag==BASIC || methodflag==BASICCEIL || methodflag==DEMIPAS)
0062     M=10;
0063 elseif(methodflag==MIS2 || methodflag==FIGURES)
0064     M=2;
0065 else
0066     M=5;
0067 end
0068
0069 if(methodflag==OPTIMAL || methodflag==OPTIMALCOMP || methodflag==OPTIMALCOMPFLOOR || methodflag==OPTIMALFLOOR ...
0070    || methodflag==DERAF || methodflag==DERAFFLOOR)
0071     AVECTSTAR=YES;
0072 else
0073     AVECTSTAR=NO;
0074 end
0075
0076 d=length(X0); %la dimension de X
0077
0078 global evaluations; % pour compter le nombre d'évaluations de la fonction flux
0079 global evaluationsd; % idem pour flux_prime_star
0080 evaluations=0;
0081 evaluationsd=0;
0082
0083 evalfin=0;
0084 evaltot=0;
0085 evalfind=0;
0086 evaltotd=0;
0087
0088 stepp=(Tfin-Tin)/N1;
0089 T=(Tin:stepp:Tfin)';
0090 if(debug==YES) %pour les affichages
0091     Told=T;
0092 end
0093 ALLGOOD=NO; % la condition d'arret
0094 it=0; %compteur d'iterations
0095
0096
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098 % Chargement des valeurs exactes    %
0099 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0100
0101 exactXflag=NO; % si l'on a la solution exacte pour X avec la c.i. X(Tin)=X0
0102 exactXciflag=NO; % si l'on a la solution exacte pour X pour toutes les c.i. possibles
0103 exactpsiflag=NO; % si l'on a la solution exacte pour psi
0104 PDE=NO; % si on est en train de resoudre une semi-discretisation d'une PDE
0105
0106
0107 if(isexact(flux,X0,Tin)) %isexact est une subfunction
0108     exactXflag=YES;
0109 end
0110
0111
0112
0113 if(strcmp(flux,'c'))
0114     exactXflag=YES;
0115     exactpsiflag=YES;
```

```
0116 end
0117
0118 if(strcmp(flux,'a') || strcmp(flux,'b'))
0119     exactXflag=YES;
0120     exactXciflag=YES;
0121     exactpsiflag=YES;
0122 end
0123
0124 if(strcmp(flux,'h') || strcmp(flux,'j') || strcmp(flux, 'burgers_upwind') || ...
0125    trcmp(flux, 'chaleur') || strcmp(flux, 'chaleurblowup'))
0126     PDE=YES;
0127 end
0128
0129
0130 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0131 % Chargement de la méthode          %
0132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0133
0134
0135 if(PDE==YES)
0136     odemethod='eulerp';
0137 else
0138     odemethod='RK';
0139 end
0140
0141 if(strcmp(flux,'chaleurblowup'))
0142     odemethod='RK';
0143 end
0144
0145
0146
0147 if(strcmp(odemethod,'heun'))
0148     p=2;
0149 elseif(strcmp(odemethod,'RK'))
0150     p=5;
0151 elseif(strcmp(odemethod,'eulerp'))
0152     p=1;
0153 end
0154
0155
0156
0157 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0158 % Chargement des dérivées           %
0159 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0160
0161 flux_prime_star=[flux,'_prime_star'];
0162 g_prime=[g,'_prime'];
0163
0164
0165
0166 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0167 % Plot des solutions finales        %
0168 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0169
0170 if(debug==YES)
0171     disp('');
0172     disp('======================');
0173     disp('MSTZ');
0174     disp('======================');
0175     figure(1);
0176     clf;
0177     stepi=(Tfin-Tin)/1000;
0178     Tplot=(Tin:stepi:Tfin)';
0179     if(exactXflag==YES)
0180     vraiX=zeros(length(Tplot),d);
0181     vraiX=exact(flux, Tplot, Tin, X0);
0182     subplot(2,3,1);
```

```
0183     hold on;
0184     plot(Tplot, vraiX(:,1),'red');
0185     else
0186     subplot(2,3,1);
0187     options=odeset('RelTol',TOL/N1, 'AbsTol', TOL/N1, 'Stats', 'on');
0188     [T45, X45]=ode45(flux, [Tin Tfin], X0, options);
0189     hold on;
0190     plot(T45,X45(:,1), 'red');
0191     end
0192
0193     if(exactpsiflag==YES)
0194     psifinal=zeros(length(Tplot),d);
0195     psifinal=exactpsi(flux, Tplot, Tfin);
0196     subplot(2,3,2);
0197     hold on;
0198     plot(Tplot, psifinal(:,1), 'black');
0199     end
0200
0201 end
0202
0203
0204 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0205 % Boucle principale                 %
0206 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0207
0208 while(ALLGOOD==NO)
0209
0210     NN=length(T);
0211     it=it+1;
0212
0213
0214     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0215     % Calcul de X                   %
0216     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0217     %c'est la valeur de référence
0218
0219     X=zeros(NN,d);
0220     X(1,:)=X0;
0221     if(methodflag~=DEMIPAS) %on calcule avec des pas entiers
0222     for i=2:NN
0223         X(i,:)=feval(odemethod,flux,T(i)-T(i-1),T(i-1),X(i-1,:));
0224     end
0225     else %on calcule avec des demi-pas
0226     for i=2:NN
0227         dt=T(i)-T(i-1);
0228         temp=feval(odemethod, flux, dt/2, T(i-1), X(i-1,:));
0229         X(i,:)=feval(odemethod, flux, dt/2, T(i-1)+dt/2, temp);
0230     end
0231     end
0232
0233     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0234     % Calcul de XX                  %
0235     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0236     %la valeur que l'on emploie pour estimer l'erreur
0237
0238     XX=zeros(NN,d);
0239     XX(1,:)=X0;
0240     if(methodflag~=DEMIPAS) %on estime l'erreur avec XX sur un mesh 2 fois plus fin
0241     for i=2:NN
0242         dt=T(i)-T(i-1);
0243         temp=feval(odemethod, flux, dt/2, T(i-1), X(i-1,:));
0244         XX(i,:)=feval(odemethod, flux, dt/2, T(i-1)+dt/2, temp);
0245     end
0246     else %on estime l'erreur avec XX sur un mesh 2 fois plus grossier
0247     for i=2:NN
0248         XX(i,:)=feval(odemethod,flux, T(i)-T(i-1),T(i-1),X(i-1,:));
0249     end
```

```
0250    end
0251
0252    evalfin=evaluations-evaltot;
0253    evaltot=evaluations;
0254
0255
0256    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0257    % Erreur locale                  %
0258    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0259
0260    %constante de Richardson
0261
0262    if(methodflag~=DEMIPAS)
0263    gamma=1/(1-(1/2)^(p));
0264    else
0265    gamma=-1/(2^p -1);
0266    end
0267
0268    %estimation de l'erreur
0269    e=gamma.*(XX-X);
0270
0271    if(debug==YES)
0272    if(exactXciflag==YES) % on calcule l'erreur exacte
0273        vraie=zeros(NN,d);
0274        vraie(1,:)=0;
0275        for i=2:NN
0276        vraie(i,:)=exact(flux,T(i),T(i-1),X(i-1,:))-X(i,:);
0277        end
0278    end
0279    end
0280
0281
0282    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0283    % Calcul de psi                  %
0284    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0285
0286    %j'applique le schema odemethod mais a l'envers, en partant de la fin
0287    psi=zeros(NN,d);
0288    psi(NN,:)=feval(g_prime,X(NN,:));
0289    i=NN-1;
0290    while i>=1
0291    psi(i,:)=feval(odemethod, 'fluxback', T(i)-T(i+1),T(i+1),psi(i+1,:),flux_prime_star,T(i),T(i+1),X(i,:),X(i+1,:));
0292    i=i-1;
0293    end
0294
0295    evalfind=evaluationsd-evaltotd;
0296    evaltotd=evaluationsd;
0297
0298    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0299    % Calcul de r                    %
0300    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0301
0302    dT=T(2:NN)-T(1:NN-1);
0303
0304    r=zeros(NN,1);
0305    for i=1:NN
0306    r(i)=e(i,:)*psi(i,:)';
0307    end
0308
0309    if(debug==YES)
0310    if(exactXciflag==YES)
0311        %attention la il faut prendre psi(t,y) !!
0312        vraipsiy=zeros(NN,d);
0313        vraipsiy=exactpsiy(flux, T, Tfin, X);
0314        vrair=zeros(NN,1);
0315        for i=1:NN
0316        vrair(i,:)=vraie(i,:)*vraipsiy(i,:)';
```

```
0317        end
0318
0319    end
0320    end
0321
0322
0323    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0324    % Condition d'arret              %
0325    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0326
0327    if(abs(sum(r))<TOL || isnan(sum(r)))
0328    ALLGOOD=YES;
0329    if(isnan(sum(r)))
0330        disp('l''estimateur d''erreur est égal à  NaN');
0331    end
0332    end
0333
0334
0335    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0336    % Calcul de rho                  %
0337    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0338
0339
0340    if(AVECTSTAR==YES && ALLGOOD==NO)
0341    rho=abs(r./[1;dT.^(p+1)]);
0342    CC=TOL^(1/p)*(rho'.^(1/(p+1))*[1;dT])^(-1/p);
0343    dTstar=zeros(NN-1,1);
0344    dTstar=CC./(rho(2:NN).^(1/(p+1)));
0345    rstar=((rho(2:NN)).*(dTstar.^p)).*dT;
0346
0347    if(methodflag==OPTIMALCOMP || methodflag==OPTIMALCOMPFLOOR)
0348        FLAG=(abs(r(2:NN))>TOL/(NN-1));
0349        ANTIFLAG=-FLAG+1;
0350        adv=ANTIFLAG'*abs(r(2:NN));
0351        CCwithadv=(abs(TOL-adv))^(1/p)*((([1; FLAG]).*rho)'.^(1/(p+1))*[1;dT])^(-1/p);
0352        dTstar=CCwithadv./(rho(2:NN).^(1/(p+1)));
0353    end
0354    end
0355
0356
0357    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0358    % Raffinement du maillage        %
0359    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0360
0361    MM=[];
0362    stepadded=0;
0363    if(ALLGOOD==NO)
0364    i=2;
0365    while i<=NN
0366
0367
0368        if(abs(r(i))>TOL/(NN-1))
0369        switch methodflag
0370         case {BASIC, DEMIPAS}
0371          mm=min(M, max(floor( (abs(r(i))/(TOL/(NN-1)))^(1/(p+1))),2));
0372         case BASICCEIL
0373          mm=min(M, ceil( (abs(r(i))/(TOL/(NN-1)))^(1/(p+1))));
0374         case {MIS2, FIGURES}
0375          mm=M;
0376         case {OPTIMAL, OPTIMALCOMP, DERAF}
0377          mm=min(M, max(ceil(dT(i-1)/dTstar(i-1)),2));
0378         case {OPTIMALCOMPFLOOR, OPTIMALFLOOR, DERAFFLOOR}
0379          mm=min(M, max(floor(dT(i-1)/dTstar(i-1)),2));
0380        end
0381        T=dividemeshm(T,mm,i-1+stepadded);
0382        stepadded=stepadded+mm-1;
0383        if(debug==YES)
```

```
0384            MM=[MM mm];
0385        end
0386        elseif( (methodflag==DERAF || methodflag==DERAFFLOOR) && ...
0387            i~=NN && dTstar(i-1)>=dT(i-1)+dT(i) && dTstar(i)>=dT(i))
0388        T=[T(1:i+stepadded-1) ;T(i+1+stepadded:NN+stepadded)];
0389        stepadded=stepadded-1;
0390        i=i+1;
0391        if(debug==YES)
0392            MM=[MM -1/2];
0393        end
0394        else
0395        if(debug==YES)
0396            MM=[MM 1];
0397        end
0398        end
0399        i=i+1;
0400    end
0401    end
0402
0403    if(debug==YES)
0404
0405    %%%%%%%%%%%%%%%%%%%%%%%%%%%
0406    % Affichage en ligne       %
0407    %%%%%%%%%%%%%%%%%%%%%%%%%%%
0408
0409    disp(' ');
0410    disp('-------------------------------------------------------');
0411        disp(['iteration no ', int2str(it)]);
0412    disp(['Nombre de pas de temps : ', int2str(NN-1)]);
0413    disp(['Dimension : ', int2str(d)]);
0414    disp(['seuil de modification du maillage : ', num2str(TOL/(NN-1),'%0.5g')]);
0415    disp(['r max : ', num2str(max(abs(r)),'%0.5g')]);
0416        disp(['seuil de sortie : ', num2str(TOL,'%0.5g')]);
0417           disp(['estimation de l erreur globale (somme des ri) : ', num2str(sum(r),'%0.5g')]);
0418
0419    vraievaleur=exact(flux,Tfin, Tin, X0);
0420    if(~isnan(vraievaleur))
0421        aaa=feval(g,vraievaleur) - feval (g,X(NN,:));
0422        disp(['g(X)-g(Xbar) : ',num2str(aaa)]);
0423    end
0424
0425    if(PDE==YES) %affichage des graphiques
0426        displayPDE;
0427    else
0428        displayODE;
0429    end
0430    Told=T; %pour les affichages de la prochaine itération
0431
0432    end
0433 end
0434
0435 if(debug==YES) %affichage final
0436    figure(1);
0437    if(PDE==YES)
0438    subplot(1,3,3);
0439    else
0440    subplot(2,3,6);
0441    end
0442
0443    hold off;
0444    plot(1,1);
0445    axis off;
0446    axis([0 1 0 1]);
0447    text(0.05,0.95,'============================================================');
0448    text(0.05,0.85,'Resolution terminee');
0449    text(0.05,0.75,['nombre d iterations : ', int2str(it)]);
0450    text(0.05,0.65,['Nombre de pas de temps : ', int2str(NN-1)]);
```

```
0451     text(0.05,0.55,['Plus grand pas de temps : ', num2str(max(Told(2:NN)-Told(1:NN-1)),'%0.5g')]);
0452     text(0.05,0.45,['Plus petit pas de temps : ', num2str(min(Told(2:NN)-Told(1:NN-1)),'%0.5g')]);
0453     text(0.05,0.35,['TOL : ', num2str(TOL)]);
0454     text(0.05,0.25,['estimation de l erreur globale (somme des ri) : ', num2str(sum(r),'%0.5g')]);
0455     if(~isnan(vraievaleur))
0456     text(0.05,0.15,['vraie erreur globale : ', num2str(aaa,'%0.5g')]);
0457     end
0458     text(0.05,0.05,['Nombre d''evaluations totales : ',num2str(evaltot)]);
0459     text(0.05,-0.05,'============================================================');
0460
0461     %saveas(gcf,name1);
0462 end
0463
0464 if methodflag==FIGURES  % fait les figures pour le rapport (validation)
0465     figure(1);
0466     subplot(1,4,1)
0467     plot(T,X(:,1));
0468     xlim([Tin Tfin]);
0469     xlabel('t');
0470     if(d==1)
0471     title('X(t)');
0472     else
0473     title('x1(t)')
0474     end
0475
0476     subplot(1,4,2);
0477     plot(T,psi(:,1));
0478     xlim([Tin Tfin]);
0479     xlabel('t');
0480     if(d==1)
0481     title('{\psi}(t)');
0482     else
0483     title('{\psi_1}(t)');
0484     end
0485
0486     subplot(1,4,3);
0487     plot(T(2:NN),e(2:NN),'o');
0488     xlim([Tin Tfin]);
0489     xlabel('t');
0490     if(d==1)
0491     title('e(t)')
0492     else
0493     title('e1(t)');
0494     end
0495
0496     subplot(1,4,4);
0497     plot(T(2:NN), dT,'o');
0498     xlabel('t');
0499     xlim([Tin Tfin]);
0500     title('taille des pas de temps');
0501     directory='../../rapport/figuresmatlab/';
0502     file=['les4_',flux,'.eps'];
0503     name=[directory,file];
0504     print('-dpsc2',name);
0505 end
0506
0507
0508 if nargout==3
0509     varargout{1}=psi;
0510 end
0511
0512 if nargout==4
0513     varargout{1}=psi;
0514     varargout{2}=e;
0515 end
0516
0517 if nargout==9
```

```
0518     varargout{1}=e;
0519     varargout{2}=sum(r);
0520     varargout{3}=evaltot;
0521     varargout{4}=evalfin;
0522     varargout{5}=evaltotd;
0523     varargout{6}=evalfind;
0524     varargout{7}=it;
0525 end
0526
0527 if nargout==10
0528     varargout{1}=psi;
0529     varargout{2}=e;
0530     varargout{3}=sum(r);
0531     varargout{4}=evaltot;
0532     varargout{5}=evalfin;
0533     varargout{6}=evaltotd;
0534     varargout{7}=evalfind;
0535     varargout{8}=it;
0536 end
0537
0538
0539
0540
0541 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0542 % Sous - fonctions                               %
0543 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0544
0545 function res=dividemeshm(T,m,k)
0546 %divise le mesh par m entre T(k) et T(k+1)
0547 res=zeros(length(T),1);
0548 res(1:k)=T(1:k);
0549 for j=1:m-1
0550     res(k+j)=T(k)+j*(T(k+1)-T(k))/m;
0551 end
0552 res(k+m:length(T)+m-1)=T(k+1:length(T));
0553
0554
0555 function res=isexact(flux, X0, Tin)
0556 % Pour alléger les test d'exactitude
0557 % donne 1 si les conditions initiales donnent une solution exacte, 0 sinon.
0558
0559 if (strcmp(flux,'h') || strcmp(flux,'burgers_upwind'))
0560     global meshX;
0561     global epsilon;
0562
0563     ul=1;
0564     ur=0;
0565     s=(ul+ur)/2;
0566     res=0;
0567     if(Tin==0)
0568     if(X0==-step(meshX,0)+1 & epsilon~=0)
0569         res=1;
0570     elseif(epsilon~=0 & X0==ur+0.5*(ul-ur)*(1-tanh( (ul-ur)*(meshX) / (4*epsilon))))
0571         res=1;
0572     elseif(X0==step(meshX,0))
0573         res=1;
0574     end
0575     end
0576 elseif(strcmp(flux,'j'))
0577     res=1;
0578 elseif(strcmp(flux,'d'))
0579     res=1;
0580 elseif(strcmp(flux,'chaleur'))
0581     global meshX;
0582     global nu;
0583     if(X0==sin(pi.*meshX) & nu==1)
0584     res=1;
```

```
0585     else
0586     res=0;
0587     end
0588 else
0589     res=0;
0590 end
```

Annexe C

Remerciements

J'aimerais remercier le professeur Alfio Quarteroni pour m'avoir accueilli dans son groupe toujours très dynamique. Je remercie Erik Burman et Paolo Zunino pour tout le temps qu'ils ont consacré à répondre à mes multiples questions. Ils ont toujours fait preuve d'une patience exemplaire et je leur en suis très reconnaissant. J'ai beaucoup appris à leur contact.

Bibliographie

[ADR02] G. Acosta, R. G. Durán et J. D. Rossi : An adaptive time step procedure for a parabolic problem with blow-up. *Computing*, 68(4):343–373, 2002.

[Ald01] R. Aldrovandi : *Special matrices of mathematical physics*. World Scientific Publishing Co. Inc., River Edge, NJ, 2001.

[BD72] Åke Björck et Germund Dahlquist : *Numerische Methoden*. R. Oldenbourg Verlag, Munich, 1972.

[BR03] Wolfgang Bangerth et Rolf Rannacher : *Adaptive finite element methods for differential equations*. Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 2003.

[BT97] Jeffrey S. Baggett et Lloyd N. Trefethen : Low-dimensional models of subcritical transition to turbulence. *Phys. Fluids*, 9(4):1043–1053, 1997.

[Bur98] Erik Burman : *Adaptive finite element methods for compressible two-phase flow*. Thèse de doctorat, Chalmers Tekniska Högskola Göteborg, 1998.

[CEES93] P. Collet, J.-P. Eckmann, H. Epstein et J. Stubbe : Analyticity for the Kuramoto-Sivashinsky equation. *Phys. D*, 67(4):321–326, 1993.

[CT65] James W. Cooley et John W. Tukey : An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19:297–301, 1965.

[Dac92] Bernard Dacorogna : *Introduction au calcul des variations*, volume 3 de *Cahiers Mathématiques de l'École Polytechnique Fédérale de Lausanne*. Presses Polytechniques et Universitaires Romandes, Lausanne, 1992.

[Dav79] Philip J. Davis : *Circulant matrices*. John Wiley & Sons, New York-Chichester-Brisbane, 1979.

[DP80] J. R. Dormand et P. J. Prince : A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.*, 6(1):19–26, 1980.

[GW99] C. Gasquet et P. Witomski : *Fourier analysis and applications*, volume 30 de *Texts in Applied Mathematics*. Springer-Verlag, New York, 1999. Filtering, numerical computation, wavelets, Translated from the French and with a preface by R. Ryan.

[HNW93] E. Hairer, S. P. Nørsett et G. Wanner : *Solving ordinary differential equations. I.* Springer Series in Computational Mathematics No 8. Springer-Verlag, Berlin, 1993.

[Hop93] Frank C. Hoppensteadt : *Analysis and simulation of chaotic systems*, volume 94 de *Applied Mathematical Sciences*. Springer-Verlag, New York, 1993.

[HW96] E. Hairer et G. Wanner : *Solving ordinary differential equations. II.* Springer Series in Computational Mathematics No 14. Springer-Verlag, Berlin, 1996.

[HYG87] Stephen M. Hammel, James A. Yorke et Celso Grebogi : Do numerical orbits of chaotic dynamical processes represent true orbits ? *J. Complexity*, 3(2):136–145, 1987.

[Kro73] Fred Krogh : On testing a subroutine for the numerical integration of ordinary differential equations. *Journal of the Association for Computing Machinery*, 20(4):545–562, 1973.

[KT76] Yoshiki Kuramoto et Toshio Tsuzuki : Persistent propagation of concentration waves in dissipative media far from thermal equilibrium. *Progress of Theor. Phys.*, 55:356–368, 1976.

[Lam91] J. D. Lambert : *Numerical methods for ordinary differential systems*. John Wiley & Sons Ltd., Chichester, 1991.

[LeV90] Randall J. LeVeque : *Numerical methods for conservation laws*. Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 1990.

[Mic96] Daniel Michelson : Stability of the Bunsen flame profiles in the Kuramoto-Sivashinsky equation. *SIAM J. Math. Anal.*, 27(3):765–781, 1996.

[MSTZ01] Kyoung-Sook Moon, Anders Szepessy, Raúl Tempone et Georgios Zouraris : Hyperbolic differential equations and adaptive numerics. *In Theory and numerics of differential equations (Durham, 2000)*, Universitext, pages 231–280. Springer, Berlin, 2001.

[MSTZ03a] Kyoung-Sook Moon, Anders Szepessy, Raúl Tempone et Georgios E. Zouraris : Convergence rates for adaptive approximation of ordinary differential equations. *Numer. Math.*, 96(1):99–129, 2003.

[MSTZ03b] Kyoung-Sook Moon, Anders Szepessy, Raúl Tempone et Georgios E. Zouraris : A variational principle for adaptive approximation of ordinary differential equations. *Numer. Math.*, 96(1):131–152, 2003.

[Ott02] Edward Ott : *Chaos in dynamical systems*. Cambridge University Press, Cambridge, 2002.

[PZ85] Y. Pomeau et S. Zaleski : The Kuramoto-Sivashinsky equation : a caricature of hydrodynamic turbulence ? *In Macroscopic modelling of turbulent flows (Nice, 1984)*, volume 230 de *Lecture Notes in Phys.*, pages 296–303. Springer, Berlin, 1985.

[QSS00] Alfio Quarteroni, Riccardo Sacco et Fausto Saleri : *Numerical mathematics*, volume 37 de *Texts in Applied Mathematics*. Springer-Verlag, New York, 2000.

[Qua00] Alfio Quarteroni : *Méthodes numériques pour le calcul scientifique*. Collection IRIS. Springer-Verlag France, Paris, 2000.

[ZC04] P. Zunino et F. Calabrò : Global existence of strong solutions of parabolic problems on partitioned domains with nonlinear conditions at the interface. Rapport technique, Ecole Polytechnique Fédérale de Lausanne, 2004.