UMONS
Université de Mons

Faculté Polytechnique

Analyse Numérique

Équation de Burger-Fisher

Rapport de Projet

TONDEUR Alice
DUPIEREUX-FETTWEIS Thomas
ZARIOH Mehdi
TOMENKO Pierre

POLYTECH MONS

Sous la direction du professeur Philippe SAUCEZ

Année Académique 2018-2019

ACADÉMIE UNIVERSITAIRE WALLONIE-BRUXELLES

Table des matières

1 Énoncé du problème

1.1 Énoncé proposé

Appartenant à la catégorie des équations aux dérivées partielles non linéaires, cette équation est utilisée pour modéliser des phénomènes relevant de domaines aussi variés que les mathématiques financières, la dynamique des gaz, le trafic routier et la physique appliquée. Elle s'écrit

$$\frac{\partial u}{\partial t} = -v(t)u^{m_1}\frac{\partial u}{\partial x} + D\frac{\partial^2 u}{\partial x^2} + \alpha(t)u(1-u^{m_2})$$

1.2 Solution analytique

Les valeurs retenues ici sont les suivantes :

$$m_1 = 2$$

$$m_2 = 3$$

$$v(t) = \frac{-2Dk}{p[b(t)]^2}$$

$$\alpha(t) = \frac{2Dk^2}{p[b(t)]^3}$$

$$b(t) = \frac{1}{p}[1 + e^{-6Dp^2k^2t}]^{\frac{1}{3}}$$

$$D = p = k = 1$$

La solution analytique de cette équation est :

$$u(x,t) = 0.5pb(t)[1 - tanh(\frac{p\xi}{2})]$$

avec

$$\xi = kx - 3pDk^2t$$

En remplaçant les variables par les valeurs retenues, on trouve :

$$b(t) = [1 + e^{-6t}]^{\frac{1}{3}}$$

$$\alpha(t) = \frac{2}{[b(t)]^3} = \frac{2}{1 + e^{-6t}}$$

$$v(t) = \frac{-2}{[b(t)]^2} = \frac{-2}{[1 + e^{-6t}]^{\frac{2}{3}}}$$

L'équation devient alors :

$$\frac{\partial u}{\partial t} = \frac{2}{[1 + e^{-6t}]^{\frac{2}{3}}}u^2\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{2}{1 + e^{-6t}}u(1-u^3)$$

Partant de la solution analytique :

$$u(x,t) = 0.5pb(t)[1 - tanh(\frac{p\xi}{2})]$$

avec

$$\xi = kx - 3pDk^2t$$

on trouve

$$\xi = x - 3t$$

et donc

$$u(x,t) = 0.5[1 + e^{-6t}]^{\frac{1}{3}}([1 - tanh(\frac{x-3t}{2})]$$

1.3 Description de la grille spatiale et temporelle

Le domaine d'étude spatial et temporel est : $-10 \leq x \leq 10$ et $0 \leq t \leq 5$. La première étape est la définition d'un maillage sur le domaine d'étude. u sera alors uniquement évalué en les valeurs de x : $x_i = x_{min} + \Delta x$ où $i : 0 \rightarrow n-1$.
Un exemple est montré ici pour définir un maillage spatial uniforme :
$x0 = -10.0$;
$xL = 10.0$;
$dx = (xL - x0)/(n-1)$;
$x = [x0 : dx : xL]'$;
Avec n le nombre de points de la grille spatiale.

1.4 Condition initiale

La condition initiale est déduite de la solution analytique. Il suffit dès lors de remplacer t par $t_0 = 0$ dans l'expression analytique pour avoir :

$$u(x,0) = 2^{\frac{-2}{3}}[1 - tanh(\frac{x}{2})]$$

1.5 Conditions aux limites

À nouveau, les conditions aux limites sont trouvées en reprenant la solution analytique, évaluée en $x_0 = -10$ et $x_L = 10$:

$$u(-10,t) = 0.5[1 + e^{-6t}]^{\frac{1}{3}}[1 - tanh(\frac{-10-3t}{2})]$$

$$u(10,t) = 0.5[1 + e^{-6t}]^{\frac{1}{3}}[1 - tanh(\frac{10-3t}{2})]$$

1.6 Description des paramètres

Ci-dessous sera présentée la description théorique des différents paramètres que nous pourrons faire varier pour observer des comportements différents en terme de précision de calcul mais aussi en ce qui concerne le temps d'exécution des 3 simulateurs que nous allons implémenter.

1.6.1 Solveurs non stiff

Les intégrateurs de Matlab que nous utiliserons pour résoudre notre EDP (transformée en système d'EDO via la méthode des lignes et des éléments finis) peuvent être conçus pour résoudre des problèmes stiff ou non. Une EDO est dite "stiff" lorsque le choix du pas d'intégration temporelle est conditionné par la stabilité de la réponse obtenue au lieu d'être conditionné par l'erreur de troncature que ce choix génère.

Nous allons d'abord décrire brièvement les intégrateurs non stiff disponibles sur Matlab.

Ode45

Cet intégrateur est bâti sur un couple RK imbriqué constitué d'une formule d'ordre quatre et d'une formule d'ordre cinq. Le choix des différents coefficients est motivé par deux objectifs. Tout d'abord réduire le coefficient du premier terme de l'erreur de troncature dans la relation du cinquième ordre. Mais aussi étendre au maximum le domaine de stabilité. Rappelons que le but est d'obtenir des domaines de stabilité les plus proches possibles pour les deux formules. Le contrôle du pas résulte du calcul de l'écart qui sépare les résultats fournis par ces deux formules.

Ode23

Cette fois, il s'agit d'un couple RK imbriqué regroupant une formule d'ordre trois et une formule d'ordre deux. Cette méthode est particulièrement efficace d'un point de vue de l'erreur de troncature et des performances de temps de calcul lorsque les tolérances sont peu élevées.

Ode113

C'est une méthode de prédiction-correction construite sur les méthodes d'Adams. La prédiction est une formule d'Adams explicite et celle de correction, d'Adams implicite. La prédiction est d'ordre k et la correction d'ordre k + 1. Matlab contient les codes implémentés jusque k = 12. Ode113 est particulièrement efficace quand les exigences de tolérance sont très sévères, puisqu'il peut tirer profit de ses formules de prédiction-correction d'ordres très élevés.

1.6.2 Solveurs stiff

La liste suivante décrit les intégrateurs présentant de meilleures performances pour les problèmes stiff.

Ode15s

Ode15s est basé sur une version améliorée des méthodes BDF classiques, appelée "NDF". Ode15s est également capable de résoudre les systèmes différentiels-algébriques.

Ode23s

C'est une méthode de Rosenbrock à deux étages. Cette méthode est L-stable, ce qui la rend particulièrement adaptée à la résolution de problèmes stiff. Au même titre que Ode23, l'ordre peu élevé de la méthode la rend particulièrement efficace lorsque les exigences de tolérance sont peu élevées.

Ode23tb

Basé sur la méthode TR-BDF2 (Bank, 1985), cette méthode à un pas peut être interprétée comme une méthode DIRK avec correction du pas d'ordre 2(3). Ode23tb convient pour les problèmes stiff lorsque les exigences de tolérance sont peu élevées grâce à ses ordres faibles, tout comme Ode23 et Ode23s.

Ode23t

Implémentant la méthode des trapèzes, elle est capable de résoudre les systèmes différentiels-algébriques, comme Ode15s.

1.6.3 Nombre de points de la grille spatiale

Une tendance générale peut être observée en modifiant le nombre de points du maillage spatial. Rendre le maillage plus fin augmente drastiquement le nombre d'opérations à effectuer, quelle que soit la méthode employée, donc a priori allonge le temps d'exécution des simulateurs. En contrepartie, l'erreur de troncature diminue avec le nombre croissant de points. Ces affirmations seront observées avec plus ou moins de véracité lors des résultats numériques par les différences finies, les éléments lagrangiens et la collocation orthogonale en fonction des autres paramètres choisis.

1.6.4 Schémas de différences finies

Paramètres qui feront l'objet de modifications uniquement dans les différences finies, les choix de schéma sont très nombreux. En effet, remplacer une dérivée spatiale par une expression ne dépendant uniquement que des évaluations u_i du maillage se construit via la manipulation des développements de Taylor. On peut alors imaginer plein de possibilités de schémas de différences finies. On les distingue par le nombre d'évaluations u_i nécessaires pour évaluer la dérivée spatiale et par la disposition de ces points. Les schémas peuvent être centrés autour de la position de la dérivée à évaluer, ou décalés à gauche ou à droite. Ils sont alors appelés "upwind". Les scripts Matlab fournis possèdent des schémas de dérivée première et seconde centrés, upwind, avec un nombre de points allant de 2 à 5.

1.6.5 Tolérances absolues et relatives

Ces deux paramètres permettent de contrôler le pas d'intégration que choisira l'intégrateur. À chaque itération, l'intégrateur se fixe un pas de longueur h pour calculer la solution suivante. Grâce à ces tolérances, il permet de calculer une erreur locale à l'étape i et si les tolérances absolues et relatives ne sont pas respectées, il revient en arrière et prend un pas h plus petit pour les satisfaire. Certains intégrateurs se montrent plus efficaces selon les valeurs de tolérance. Des tolérances sévères vont mieux correspondre à certaines configurations tandis que des tolérances plus légères permettront un temps d'exécution plus court.

1.6.6 Remarque concernant les conditions de test

Étant donné que nous avons une série de tests à effectuer concernant l'erreur commise par rapport à la solution analytique et le temps mis par le simulateur pour résoudre l'EDP, les résultats sont fort dépendants de l'ordinateur utilisé et des conditions dans lesquelles il est sollicité. Nous avons, pour l'ensemble des tests, utilisé le même ordinateur portable, muni de la version Matlab 2016. Dans la mesure du possible, les tests ont été effectués dans les mêmes conditions d'utilisation du processeur (aucune activité en arrière-plan) et de la mémoire RAM. Les valeurs chiffrées, lues sur le terminal, sont donc pertinentes uniquement dans cette configuration. Nous en retiendrons uniquement les tendances et évolutions et ne garantissons pas les mêmes valeurs d'un PC à l'autre.

2 Différences finies

2.1 Introduction

Dans cette partie de l'énoncé, notre objectif est de résoudre numériquement le problème énoncé ci-dessus à l'aide des différences finies. Nous utiliserons tout d'abord les outils les plus simples possibles comme l'intégrateur ode45, le schéma de différences finies à nombre minimal

de points et le nombre de points de grille suffisant.

Nous représenterons le graphique de $u(x,t)$ en fonction de x pour $t_i = i\Delta t$
où $\Delta t = .5$ et $i = 0, 1, ..., 10$

Afin de vérifier la qualité de notre simulation, la solution analytique sera superposée à la solution numérique. Outre cette évaluation graphique de la qualité, on programmera aussi le calcul de l'évaluation suivante de l'écart entre solution analytique et solution numérique :

$$ec = \frac{1}{n_t} \sum_{i=0}^{10} || \overline{u}_{anal}(t_i) - \overline{u}_{num}(t_i) ||$$

où n_t est le nombre d'instants de visualisation
et $\overline{u}_{anal}(t_i)$ respectivement $\overline{u}_{num}(t_i)$ vecteur de dimension n contenant les valeurs de u_{anal} (respectivement u_{num}) en tous les points de la grille à l'instant t_i

Nous discuterons ensuite de l'influence du choix des outils sur la qualité des résultats. En effet, nous avons la main prise sur le choix de l'intégrateur temporel et sur le schéma de différences finies. Nous comparerons donc les *ec* mais aussi les temps de calcul correspondant aux différents choix. Lorsque le choix de l'intégrateur sera fait, nous tiendrons compte de l'option JPattern sur les solveurs stiff (car elle n'est disponible que pour ces types d'intégrateurs). L'implémentation du JPattern est expliquée plus loin dans ce rapport. Nous mesurerons ensuite l'impact du JPattern sur notre simulation.

Finalement, lorsque toutes nos simulations seront effectuées, nous pourrons définir quels sont les choix menant aux meilleurs résultats. Nos critères seront la qualité du profil de $u(x,t)$, le temps de calcul correspondant à chaque méthode et l'erreur *ec* générée.

2.2 Rappels théoriques : méthode des lignes

De manière générale, la méthode des lignes se présente comme suit. Si l'équation à résoudre est de la forme $L(u, u_t, u_x, u_{xx}, ..., t) = 0$, alors les étapes sont les suivantes :

1. Définition d'un maillage spatial sur le domaine d'étude : les seules valeurs de x en lesquelles u sera évalué sont

$$x_i = x_{min} + i\Delta x$$
$$i : 0 \to N$$

et

$$\Delta x = \frac{x_{max} - x_{min}}{N}$$

pour un maillage uniforme.
2. Écriture de l'équation à résoudre en chaque valeur de u_i du domaine spatial :

$$L(u_i, (u_t)_i, (u_x)_i, (u_{xx})_i, ..., t) = 0$$

avec

$$i = 1, ..., N$$

Cela permet l'écriture d'un système d'EDP avec les dérivées spatiales et temporelles évaluées aux noeuds du maillage.

3. Remplacement des dérivées spatiales par les schémas de différences finies

Les dérivées spatiales sont substituées par leurs équivalents en différences finies. Parmi tous les schémas existants, voici le 2 points upwind pour la dérivée première et le 3 points centrés pour la dérivée seconde :

$$(u_x)_i = \frac{u_i - u_{i-1}}{\Delta x}$$

$$(u_{xx})_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

On a alors un système d'équations différentielles ordinaires en les inconnues $u_i(t)$ que l'on peut résoudre avec les intégrateurs temporels de Matlab.

4. La condition initiale qui accompagne l'équation aux dérivées partielles fournit la valeur de chacune des fonctions inconnues $u_1, ..., u_N$ à l'instant initial. L'intégrateur ayant besoin de ce vecteur de conditions initiales, il est désormais capable de résoudre le système d'EDO. Les conditions aux limites font l'objet d'une implémentation particulière qui sera introduite dans la section prévue à cet effet.

C'est donc sur cette base théorique que nous avons implémenté notre code.

2.3 Implémentation

Cette section va décrire les étapes de l'implémentation qui nous ont permis de concevoir notre simulateur par les différences finies. Les programmes sont disponibles en annexe de ce rapport.

2.3.1 Programme principal : DiffFinies.m

Les différentes étapes de résolution des méthodes des lignes sont les suivantes :
La première étape est de définir un maillage sur le domaine d'étude sous forme d'un vecteur colonne après avoir fixé le nombre de points n du maillage.

Après cela, il s'agira de définir les conditions initiales qui fournissent le vecteur $u_1, ..., u_n$ à l'instant initial.
L'étape suivante sera alors de remplacer les dérivées spatiales par des approximations algébriques dans le but d'obtenir un système d'équations différentielles ordinaires en les fonctions du temps inconnues $u_i(t)$. Cela se fait en choisissant un schéma de différences finies qui évalue numériquement les dérivées premières et secondes de l'équation. Ces schémas sont présentés sous la forme de matrices carrées D1 et D2.
Par la suite, il faudra déterminer les paramètres de l'intégration temporelle c'est-à-dire les instants auxquels on souhaite visualiser la solution ainsi que les options utilisées par l'intégrateur temporel. Nous pourrons choisir d'activer l'option JPattern (décrite quelques sections plus bas) pour la résolution numérique.

On pourra alors intégrer notre système avec l'intégrateur temporel de notre choix en lui donnant les options, le système d'EDO à résoudre via le sous-programme BurgerFisherDiff.m, les instants de visualisation et le vecteur de CI. La dernière partie du code est consacrée à l'affichage de la solution analytique et numérique qui permettra d'évaluer la qualité de la solution numérique ainsi qu'au calcul et à l'affichage de *ec.*

2.3.2 Sous-programme de calcul du système d'EDO : BurgerFisherDiff.m

BurgerFisherDiff.m est un sous programme appelé dans le main DiffFinies.m. Dans ce sous-programme, u_t désigne la sortie et BurgerFisherDiff le nom de la fonction. Dans les paramètres,

t représente l'instant courant et u le vecteur des variables dépendantes à cet instant.

Les conditions aux limites sont définies à partir de la solution analytique (u_anal). Elles sont de type Dirichlet donc elles suppriment a priori deux lignes du système d'EDO classique. Les matrices D_1 et D_2 sont cependant programmées sans faire intervenir des noeuds fictifs au bord du domaine quand le schéma a besoin de noeuds extérieurs au maillage. Le système doit donc rester de dimension n. Nous pouvons déjà fixer la valeur de $u(1) = u(x_0, t)$ et $u(n) = u(x_L, t)$ à l'instant d'intégration pour la suite des opérations.

L'étape suivante est la construction du système d'EDO.

$$u_t = a.u2.u_x + u_{xx} + b(u - u4)$$

avec

$$a = \frac{2}{[1 + e^{-6t}]^{\frac{2}{3}}}$$
$$u2 = u^2$$
$$u_x = \frac{\partial u}{\partial x}$$
$$u_{xx} = \frac{\partial^2 u}{\partial x^2}$$
$$b = \frac{2}{1 + e^{-6t}}$$
$$u4 = u^4$$

qui correspond à l'équation de Burger Fisher à résoudre :

$$\frac{\partial u}{\partial t} = \frac{2}{[1 + e^{-6t}]^{\frac{2}{3}}} u^2 \frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{2}{1 + e^{-6t}} u(1 - u^3)$$

Finalement, on fixe les CL $ut(1) = 0$ et $ut(n) = 0$ pour indiquer à l'intégrateur que toutes les lignes, excepté la première et la dernière ligne, font l'objet de l'intégration temporelle voulue. On compense cette manipulation qui introduit une erreur aux limites par un rapatriement des conditions aux limites après intégration complète dans le programme principal.

2.3.3 Autres sous-programmes

Différents sous-programmes sont également présents dans cette partie du code comme u_anal.m qui contient la solution analytique calculée théoriquement. Il y a également ec.m qui nous est utile pour calculer ec dans le main. On trouve également les définitions des différents types d'approximation des dérivées premières et secondes de l'équation.

2.3.4 JPattern

Le JPattern, ou encore "Jacobian sparsity pattern" est une option des intégrateurs de Matlab qui permet d'améliorer les performances en terme de temps de calcul sur les intégrateurs stiff. Ceux-ci résolvant des systèmes non linéaires de manière itérative, ils peuvent avoir besoin d'évaluer à chaque itération la matrice Jacobienne J de manière numérique dans le cadre de la méthode de Newton généralisée pour les systèmes d'équations non linéaires.

Qui plus est, une EDP transformée en système d'EDO via les différences finies génère une matrice de très grande taille avec peu d'éléments non nuls et par conséquent, une matrice Jacobienne creuse également. Numériquement, il est donc intéressant de spécifier à l'intégrateur la

localisation des éléments non nuls de la matrice Jacobienne qu'il devra calculer numériquement pour éviter toute opération inutile sur les autres éléments de la matrice J égaux à 0.

L'option JPattern a besoin d'une matrice creuse, de la même dimension que la matrice J, qui contient aux emplacements des éléments non nuls des 1. Cela permettra à l'intégrateur de repérer les endroits où l'évaluation numérique doit avoir lieu dans la matrice J et placer des 0 ailleurs.

Le JPattern a donc une utilité double. Théoriquement, elle permet de pouvoir appliquer la méthode itérative de Newton généralisée aux systèmes non linéaires qui est une méthode de convergence quadratique (généralement). Numériquement, fournir la matrice indiquant les emplacements non nuls de la matrice J permet d'économiser du temps de calcul.

Il faut maintenant trouver les emplacements non nuls de la matrice J relative à notre système d'EDO. Les détails du raisonnement pour y arriver est disposé ci-dessous :

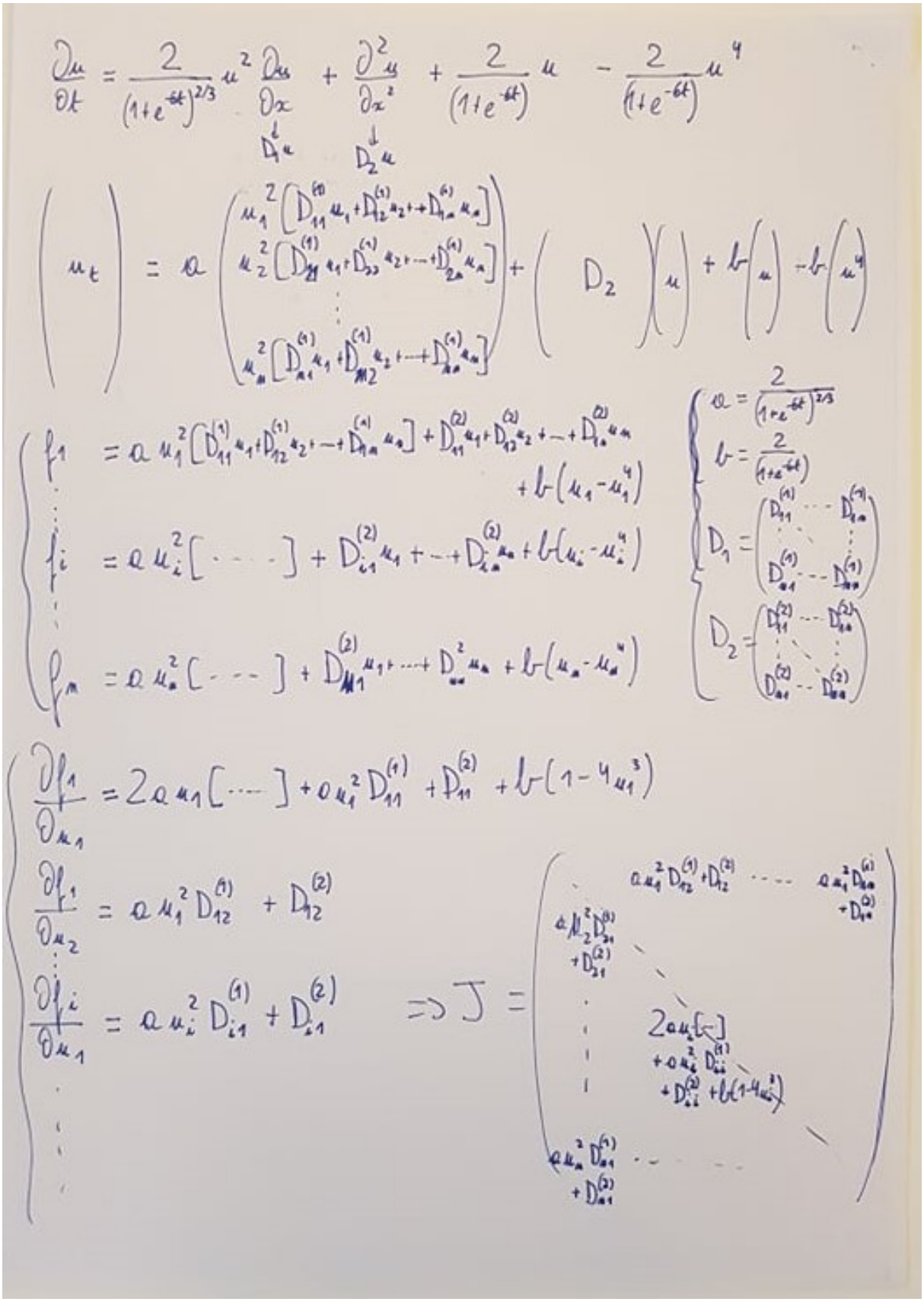


FIGURE 1: Calcul de la matrice Jacobienne du problème

Les éléments hors diagonaux (i,j) avec $i \neq j$ ont comme expression :

$$J(i,j) = au_i^2 D_{ij}^{(1)} + D_{ij}^{(2)}$$

Si $D_{ij}^{(1)} = D_{ij}^{(2)} = 0$, alors l'élément (i,j) de la matrice Jacobienne est nul.

Les éléments diagonaux (i,i) ont comme expression :

$$J(i,i) = 2au_i[\sum_j D_{ij}^{(1)} u_j] + au_i^2 D_{ii}^{(1)} + D_{ii}^{(2)} + b(1 - 4u_i^3)$$

Pour un élément diagonal, il est très rarement nul puisqu'il y a la présence du terme indépendant b (le coefficient temporel b est non nul dans notre domaine d'étude).

La matrice à fournir pour permettre l'exécution du JPattern est une matrice creuse contenant des éléments non nuls aux endroits où J est non nul. Il est possible de retrouver cette matrice en additionnant simplement la matrice $spones(D_1)$, $spones(D_2)$ et la matrice identité ensemble (l'instruction $spones(A)$ remplace tous les éléments non nuls de A par 1). En effet, sur les éléments hors diagonaux, si $D_{ij}^{(1)} = D_{ij}^{(2)} = 0$, alors la matrice du JPattern à cet élément sera aussi nul. Pour les éléments diagonaux, la matrice identité assure la présence d'un terme non nul à ces endroits-là. Le résultat de cette addition est rendu creux et on y applique une dernière fois l'instruction $spones()$.

Notons que les lignes de codes suivantes dans le programme principal DiffFinies.m :

$$J = [spones(eye(n)) + spones(D1) + spones(D2)];$$

$$J = spones(J);$$

$$J = sparse(J);$$

empêchent la matrice Jacobienne d'être nulle dans certains cas "pathologiques" et induisent donc des calculs inutiles. La condition rigoureuse pour qu'un élément hors diagonal soit nul est $au_i^2 D_{ij}^{(1)} + D_{ij}^{(2)} = 0$. il est également possible de rendre un élément diagonal nul si le coefficient b est égal à une certaine expression tirée de la relation précédente exprimant $J(i,i) = 0$. Vu la structure des matrices D_1 et D_2 comportant la majorité de ses éléments sur les diagonales, les cas pathologiques où ces contributions $J(i,i)$ et $J(i,j)$ sont nulles autrement que par les conditions citées précédemment sont extrêmement rares et n'interviendront pas sur le résultat final en terme de performance du simulateur.

2.4 Résultats numériques

Dans un premier temps, nous avons choisi un nombre de points égal à 250, l'intégrateur ode45, sans JPattern, avec D1=two_point_upwind_D1(x,1) et D2=three_point_centered_D2(x) et des tolérances absolues et relatives à 10^{-3}. La solution numérique superposée sur l'analytique ainsi que le graphe des erreurs sont montrés ci-dessous :

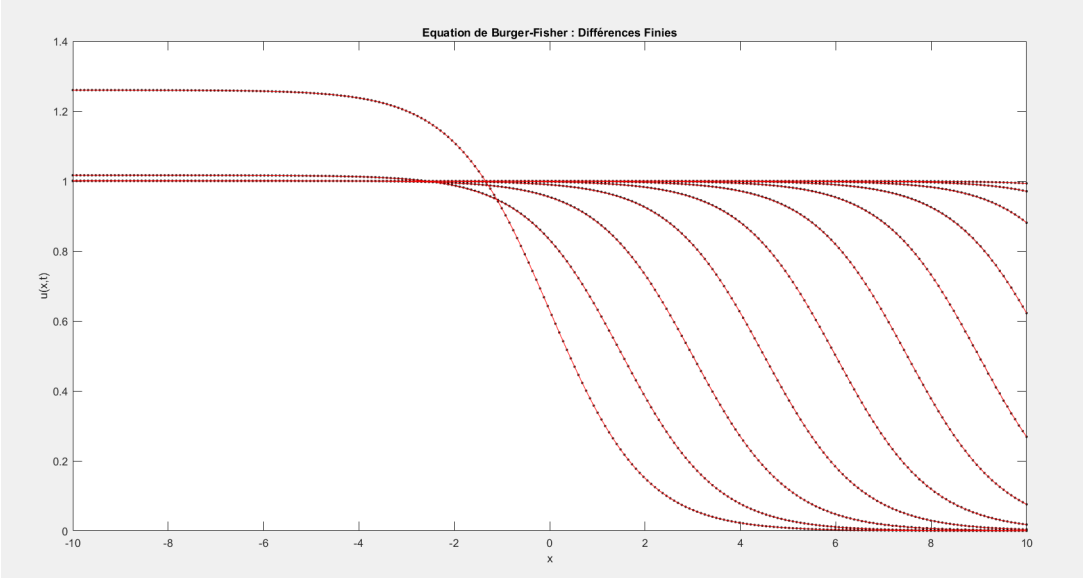


FIGURE 2: Solution numérique VS Solution analytique : Différences finies

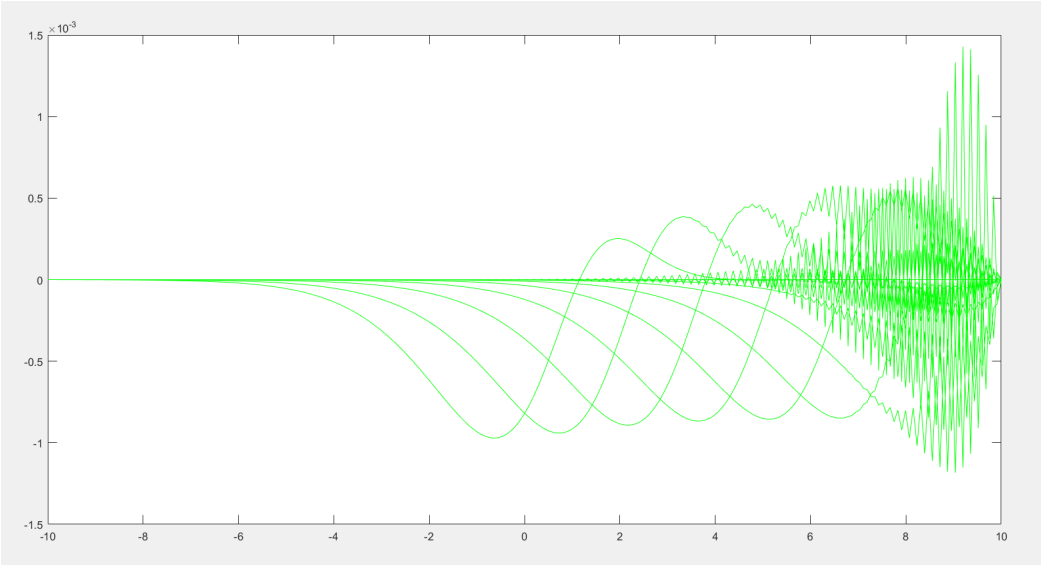


FIGURE 3: Erreurs pour chaque instant de visualisation : Différences finies

Le simulateur produit une solution numérique qui se superpose parfaitement à la solution analytique avec les paramètres les plus simples possibles. Nous pouvons aussi voir que les erreurs sont concentrées sur le bord droit du domaine dans cette configuration.

Ce simulateur fonctionnant dans sa forme la plus basique, nous pouvons maintenant discuter de l'influence des paramètres. Nous avons la possibilité de tester un nombre impressionnant de combinaisons puisque nous pouvons agir sur un total de 7 paramètres : le nombre de points, l'intégrateur, le JPattern, D1, D2, la tolérance absolue et la tolérance relative. Un tableau de dimension 7 n'étant pas réalisable et surtout peu pertinent, nous nous contenterons de faire varier certains paramètres et laisser les autres constants et en tirer un maximum de conclusions sur les différentes séries de tests.

Une première série de tests compare le choix de l'intégrateur et le choix des tolérances absolues

et relatives (que nous supposerons égales pour l'ensemble des tests). Les données sont reprises ci-dessous :

	nombre de points = 250 ; D1=five_point_biased_upwind_D1(x,1) ; D2=five_point_centered_D2(x) ; Jpattern = non								
	Tolérances	10^-3		10^-5		10^-7		10^-9	
	Intégrateurs	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err
non-stiff	ode45	0,38	0,0019	0,38	2,11E-05	0,38	2,02E-07	0,58	1,77E-07
	ode23	0,30	0,0032	0,29	3,07E-05	0,29	4,07E-07	0,51	1,78E-07
	ode113	0,48	0,0092	0,45	5,44E-06	0,46	4,84E-07	0,48	1,77E-07
stiff	ode15s	0,26	0,0067	0,27	6,60E-05	0,32	4,97E-07	0,35	1,72E-07
	ode23s	0,78	0,0084	3,8	3,03E-04	19	1,17E-05	90,6	5,34E-07
	ode23tb	0,23	0,0061	0,29	2,50E-04	0,60	1,17E-05	1,95	6,06E-07
	ode23t	0,25	0,0064	0,27	2,72E-04	0,43	1,61E-05	1,1	8,44E-07

FIGURE 4: Comparaisons intégrateurs VS tolérances

Si nous regardons d'abord l'influence de la tolérance sur l'erreur et le temps, la durée d'exécution semble ne pas varier pour la plupart des intégrateurs lorsque l'on met des tolérances jusque 10^{-7} (seul ode23s montre un signe clair de faiblesse à cet égard, et ode23tb, ode23t commencent à en montrer), et l'erreur est diminuée d'un facteur 10^{-2} par saut de 10^{-2} de tolérance. C'est en poussant le simulateur jusqu'à un niveau de tolérance de 10^{-9} que tous les intégrateurs, à l'exception d'ode113 et d'ode15s, ralentissent fortement. Le gain en précision n'est à ce moment-là plus suffisamment important pour justifier une tolérance si sévère. Elle engendre un temps de calcul bien plus long. Nous fixerons donc pour la suite des tests une tolérance de 10^{-7}.

Concernant les intégrateurs, ode113 et ode15s sont les plus adaptés aux tolérances sévères puisqu'ils sont les seuls à garder une rapidité décente pour 10^{-9} comparé au seuil précédent. Pour ode113, cela a été pressenti lors de la description des intégrateurs dans le chapitre précédent du rapport. À l'inverse, les résultats en terme de rapidité sont les meilleurs pour une tolérance plus douce avec ode23, ode15s, ode23tb et ode23t. En effet, leurs ordres faibles permettent de meilleurs performances en terme de temps de calcul pour des niveaux peu contraignants de tolérance.

De manière générale, pour une tolérance de 10^{-7}, l'intégrateur non-stiff ayant le meilleur compromis entre l'erreur et le temps de calcul est ode23, tandis que l'intégrateur stiff le plus performant dans sa catégorie est ode15s. C'est sur base de ces deux intégrateurs et de la tolérance 10^{-7} que nous poursuivrons nos tests.

La deuxième série de tests met en lumière les effets des schémas de différences finies sur la qualité numérique de la solution ainsi que le temps d'exécution pour ode23 et ode15s avec 250 points et une tolérance de 10^{-7}. Nous verrons également l'influence du JPattern sur le résultat final lorsque nous l'activerons sur ode15s :

			nombre de points = 250 ; tolérances = 10-7											
			ode23				ode15s							
			Sans Jpattern				Sans Jpattern				Avec Jpattern			
D1/D2			5pts centrés		3 pts centrés		5pts centrés		3 pts centrés		5pts centrés		3 pts centrés	
			temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err
5 pts	biased upwind	1	0,29	4,0727E-07	0,28	1,0076E-04	0,34	4,9650E-07	0,32	1,0061E-04	0,29	4,9650E-07	0,29	1,0061E-04
		-1	0,31	2,8557E-07	0,28	1,0077E-04	0,32	5,1402E-07	0,31	1,0061E-04	0,29	5,1402E-07	0,29	1,0061E-04
	centré		0,30	3,9459E-07	0,27	1,0071E-04	0,31	6,1685E-07	0,31	1,0056E-04	0,29	6,1685E-07	0,29	1,0056E-04
4 pts	biased upwind	1	0,29	2,5556E-06	0,27	1,0143E-04	0,31	2,3156E-06	0,31	1,0128E-04	0,29	2,3156E-06	0,29	1,0128E-04
		-1	0,30	2,5027E-06	0,27	1,0006E-04	0,31	2,7167E-06	0,32	9,9909E-05	0,29	2,7167E-06	0,29	9,9909E-05
	upwind	1	0,29	7,1435E-06	0,27	9,9044E-05	0,31	7,3390E-06	0,32	9,8854E-05	0,29	7,3390E-06	0,29	9,8853E-05
		-1	0,32	7,5528E-06	0,29	1,0317E-04	0,31	7,2840E-06	0,32	1,0298E-04	0,29	7,2840E-06	0,30	1,0298E-04
3 pts	upwind	1	0,29	1,4783E-04	0,27	1,6267E-04	0,31	1,4757E-04	0,31	1,6238E-04	0,29	1,4757E-04	0,29	1,6238E-04
		-1	0,31	1,5254E-04	0,28	1,6498E-04	0,31	1,5230E-04	0,31	1,6470E-04	0,29	1,5230E-04	0,29	1,6470E-04
	centré		0,29	7,5257E-05	0,27	1,2965E-04	0,31	7,5514E-05	0,31	1,2962E-04	0,29	7,5514E-05	0,28	1,2962E-04
2 pts	upwind	1	0,30	3,5000E-03	0,27	3,5000E-03	0,31	3,5000E-03	0,31	3,5000E-03	0,29	3,5000E-03	0,28	3,5000E-03
		-1	0,30	3,5000E-03	0,27	3,5000E-03	0,32	3,5000E-03	0,31	3,5000E-03	0,29	3,5000E-03	0,28	3,5000E-03

FIGURE 5: Comparaisons D1 VS D2 VS JPattern

Regardons dans un premier temps l'influence du schéma de différences finies pour la dérivée seconde D2 sur la qualité de la solution. Quelle que soit la configuration, passer d'un schéma 3 points centrés à 5 points centrés augmente le temps de calcul mais réduit l'erreur de la solution numérique d'une manière non négligeable pour les schémas de dérivée première D1 à minimum 4 points. L'effet bénéfique sur l'erreur s'estompe pour les schémas D1 à 2 ou 3 points, voire devient totalement négligeable. La perte de temps pour atteindre un résultat bien plus précis est négligeable également : passer de 0.28 s à 0.3 s en moyenne pour ode23 est une augmentation très minime. Pour aller plus loin, nous pourrions également imaginer prendre un nombre plus important de points pour marquer davantage le temps d'exécution. À titre d'exemple, pour $n = 750$ avec ode23, D1 = 4 points upwind (1) :

— $t = 3.0077$, $ec = 1.9338e - 05$ avec D2 = 3 points centrés

— $t = 4.0507$, $ec = 8.1669e - 07$ avec D2 = 5 points centrés

L'évolution en terme d'erreur est toujours d'un facteur d'environ 10^{-1} mais le temps d'exécution augmente de 33%. Il est donc clair que le schéma D2 à 5 points centrés est plus précis, au détriment d'un temps de calcul plus long. Mais étant donné que nous considérons 250 points, le temps de calcul n'est pas un facteur qui varie beaucoup. Il est évident qu'en fonction du nombre de points du maillage spatial, choisir le bon D2 pour trouver un compromis sera vital.

Concernant la discussion autour du choix de D1, elle sera plus compliquée à faire vu le nombre de choix disponibles. Les différences sont négligeables lorsque D2 est à 3 points centrés, l'erreur ec tournant autour d'une valeur de 10^{-4} quel que soit le schéma D1 choisi et quel que soit l'intégrateur. L'erreur est probablement produite exclusivement à cause du choix de D2 peu sophistiqué, rendant la contribution de D1 aux erreurs trop petite pour qu'elle soit observable. Regardons plutôt l'effet de D1 lorsque D2 est à 5 points centrés.

Pour les schémas D1 à 4 points, les schémas biaised upwind sont 3 fois plus précis selon nos chiffres en terme d'erreur que leurs équivalents upwind. À nombre de points équivalents, le centré à 5 points se montre moins efficace que les biaised upwind tandis que le centré à 3 points est plus efficace que les upwind à 3 points. Concernant la durée d'exécution du simulateur, les temps sont très proches quel que soit le D1 choisi. Pour notre étude à 250 points, nous pourrons donc prendre sans aucun problème les D1 et D2 les plus sophistiqués sans compromettre le temps de calcul.

Contrairement au choix de D2, le choix de D1 pour un nombre plus important de points ne va pas fondamentalement changer le temps d'exécution. Que le schéma D1 soit à 2,3,4,5 points, le temps moyen est de 4 secondes pour ode23 à 750 points.

Intéressons-nous maintenant au JPattern pour l'intégrateur stiff ode15s. Les erreurs sont indépendantes de la présence du JPattern. Son activation ne rendra pas le simulateur plus précis. En revanche, pour un même résultat numérique, le temps d'exécution est diminué quelle que soit la configuration pour ode15s. Cela provient du fait que le JPattern n'optimise que les calculs effectués par l'intégrateur temporel, via les emplacements non nuls de la matrice Jacobienne, sans modifier la structure du système d'EDO à résoudre. Le gain de temps offert par le JPattern n'est pas très notable pour nos tests à 250 points, mais il n'empêche que l'activation du JPattern n'a pas d'inconvénient à être utilisé, mis à part la réflexion théorique pour implémenter le JPattern.

Le gain de temps est considérable si l'on prend un nombre de points démesurément grand. Pour 2000 points, D1 = 5 points biaised upwind (1), D2 = 5 points centrés, tolérances à 10^{-7} et deux intégrateurs :

— ode15s sans JPattern : $t = 6.7682$, $ec = 1.6122e - 06$

— ode15s avec JPattern : $t = 0.4885$, $ec = 1.6122e - 06$

— ode23s sans JPattern : $t = 3077.3$, $ec = 5.7062e - 06$

— ode23s avec JPattern : $t = 27.4821$, $ec = 5.8101e - 06$

On remarque très clairement la puissance du JPattern dans ces situations.

Enfin, nous pouvons regarder plus en profondeur où sont localisées les erreurs. Les graphes des erreurs sont présentés en annexe pour l'intégrateur ode15s avec JPattern pour 250 points et tolérances de 10^{-7}. Toutes les combinaisons de D1 et D2 sont affichées. Reprenons l'un de ceux-ci en l'alignant avec la solution numérique :

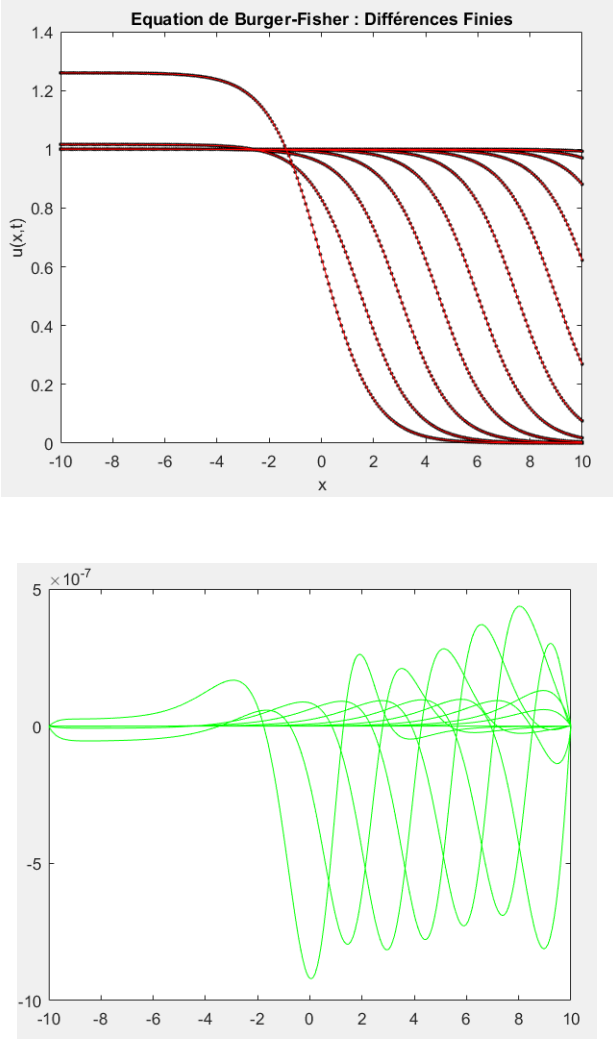


FIGURE 6: D1 = 4 points biased upwind(1) ; D2 = 5 points centrés

Les erreurs apparaissent spatialement à l'endroit où la fonction $u(x,t)$ devient plus raide, c'est-à-dire à partir de $x = 0$. Cela semble logique puisque localement, la fonction est plus difficile à intégrer. Pour palier à ce problème, une piste serait de prendre un maillage non uniforme, pour permettre qu'il soit plus dense sur la partie droite du domaine et ainsi réduire les erreurs à des endroits "stratégiques". Cette tendance d'apparition d'erreurs à partir de $x = 0$ se confirme sur tous les graphiques d'erreurs en annexe. Notons également que les graphes d'erreurs pour D2 à 3 points centrés sont similaires, voire identiques sur les 7 premiers schémas de D1. Le schéma D2 peu précis génère bien plus d'erreurs que les D1 qui lui sont associés, donc la contribution des erreurs amenées par D1 est écrasée par celles de D2.

2.5 Conclusions intermédiaires

Toutes les combinaisons possibles ne sont pas représentables. Nous nous contenterons d'en dégager les tendances principales pour le simulateur via les différences finies :

- Certains intégrateurs sont plus efficaces selon la tolérance choisie ;
- La tolérance maximale qui donne des résultats satisfaisants en terme de précision sans détériorer le temps de calcul est 10^{-7} ;
- Plus D2 est sophistiqué, plus la solution est précise mais la durée d'exécution est allongée : pour 250 points, D2 à 5 points centrés est le meilleur ;
- Le choix de D1 n'a pas beaucoup d'importance quand D2 est à 3 points centrés : D2 génère alors trop d'erreurs que D1 n'est pas capable de compenser ;
- Le JPattern est une option n'offrant pas de désavantage en terme d'erreur et réduit grandement le temps d'exécution pour des systèmes de grande taille ;
- Les erreurs sont localisées aux endroits où la pente de la fonction est plus élevée.

Si nous devions choisir le simulateur idéal, nous le paramétrerions comme ceci (pour 250 points) :

- Tolérances : 10^{-7}
- Nombre de points : 250
- Intégrateur : ode15s
- D1 : 5 points biaised upwind (1 ou -1)
- D2 : 5 points centrés
- JPattern : activé

3 Éléments Lagrangiens

3.1 Rappel théorique

Supposons traiter une équation aux dérivées partielles pouvant s'écrire sous cette forme :

$$u_t = F(u)$$

$$u_t = \alpha_0 u + \alpha_1 u_x + \alpha_2 u_{xx} + src(u)$$

Avec $src(u)$ le terme source reprenant tous les termes non linéaires de l'EDP.

En prenant comme modèle mathématique $\hat{u} = \sum_{k=1}^{E} u_k(t) N_k(x)$ dans le cadre de la théorie des éléments finis, les résidus pondérés, en négligeant les conditions aux limites dans un premier temps, peuvent s'écrire comme ceci :

$$\int_{\Omega(x)} W_l F\left(\sum_{k=1}^{E} u_{kt}(t) N_k(x)\right) dx = \alpha_0 \int_{\Omega(x)} W_l \hat{u} dx + \alpha_1 \int_{\Omega(x)} W_l \hat{u}_x dx + \alpha_2 \int_{\Omega(x)} W_l \hat{u}_{xx} dx + \int_{\Omega(x)} W_l src(\hat{u}) dx$$

Résidus pondérés sur le domaine Ω

Un élément lagrangien s'exprime, en coordonnée réduite ξ, par (avec a et b les extrémités gauche et droite de l'élément considéré) :

$$\hat{u}(\xi, t) = u_a(t) N_1(\xi) + u_b(t) N_2(\xi)$$

La méthode de Galerkin sur les éléments Lagrangiens impose de remplacer les fonctions de pondération W_l par les fonctions de base N_l du modèle mathématique lagrangien. Ceci étant fait, l'expression des résidus pondérés sur tout le domaine Ω forme un système d'EDO. Il reste donc à déterminer de quelle manière un élément lagrangien interviendra dans la construction de ce système d'EDO.

Un élément ne contribue qu'à deux lignes du système complet. Puisque l'expression des résidus pondérés fait intervenir le produit $N_l\hat{u}$ dans tous les termes envisagés, les seules contributions non nulles sont celles où la fonction de base est non nulle. Sur un élément, uniquement deux fonctions de base sont différentes de 0 donc nous avons une matrice-pavé de dimension 2x2 pour le terme de dérivée temporelle et les termes linéaires de dérivée spatiale d'ordre 0, 1 et 2. Ces matrices M, D_0, D_1, D_2 assemblées pour un système complet de taille n, sont disponibles dans les scripts Matlab fournis.

L'évaluation de l'intégrale du terme source est plus délicate car nous sommes contraints de la déterminer via l'intégration numérique. La formule de Gauss :

$$\int_a^b f(x)dx = \sum_k w_k f(x_k)$$

permet d'évaluer numériquement l'intégrale en se fixant un nombre de points. L'obtention des abscisses x_k et des poids w_k optimaux pour la formule se détermine dans les codes fournis.

Ceci étant acquis, la contribution du terme source pour un élément est un vecteur de dimension 2 (l'indice i indique une intégration numérique) :

$$\frac{h_k}{2}\begin{pmatrix} < N_1 src(\hat{u}) >_i \\ < N_2 src(\hat{u}) >_i \end{pmatrix}$$

L'assemblage de toutes les contributions des éléments nous permet de reconstituer le vecteur $\overline{f}_{NL}(\overline{u})$ qui s'ajoutera aux autres termes de dérivation d'ordre 0, 1 et 2 pour former le système d'EDO suivant :

$$M\overline{u_t} = \alpha_0 D_0 \overline{u} + \alpha_1 D_1 \overline{u} + \alpha_2 D_2 \overline{u} + \overline{f}_{NL}(\overline{u})$$

Il reste à prendre en compte les conditions aux limites. Pour cela, nous allons enfin exprimer les résidus pondérés sur le bord du domaine Γ. Soient les CL qui peuvent être mises sous la forme $M\hat{u} + r = 0$ le long de Γ. En développant les résidus pondérés :

$$\int_\Omega W_l ... d\Omega + \int_\Gamma \overline{W}_l (M\hat{u} + r) d\Gamma = 0$$

$$... - \int_\Gamma N_l (M\hat{u} + r) d\Gamma = 0$$

$$... - \{N_l(M\hat{u} + r)\}_{x_0} - \{N_l(M\hat{u} + r)\}_{x_L} = 0$$

Le terme évalué en x_0 (concernant donc le premier élément du domaine) n'affectera que la première ligne du système puisque seule la fonction de base N_1 est non nulle en x_0, et le terme évalué en x_L (concernant le dernier élément du domaine) ajoutera une contribution sur la dernière ligne du système uniquement car c'est la fonction de base N_2 qui est non nulle en x_L.

Nos conditions sont de type Dirichlet. Il faudra donc ajouter $-(u_1 - u(x_0, t))$ et $-(u_N - u(x_L, t))$ respectivement sur la première et dernière ligne du système d'EDO.

3.2 Implémentation

Notre équation

$$\frac{\partial u}{\partial t} = \frac{2}{[1+e^{-6t}]^{\frac{2}{3}}} u^2 \frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{2}{1+e^{-6t}} u(1-u^3)$$

montre deux termes linéaires de dérivée spatiale d'ordre 0 et d'ordre 2. Nous devrons donc implémenter la matrice D_0 et D_2. Il y a également un terme source, une partie en $u^2 u_x$ et une autre en u^4.

Les CL, CI et discrétisation temporelle sont toujours les mêmes que celles détaillées dans l'énoncé du problème. Le simulateur se découpe en 3 programmes, remplissant chacun un rôle particulier. Ces 3 programmes sont fournis en annexe.

3.2.1 Programme principal : Lagrange.m

Après avoir défini les variables du problème permettant de construire la grille spatiale selon le nombre de points n (qui générera par la suite un système d'EDO de dimension n), nous déclarons les matrices de dérivations spatiales qui vont intervenir dans notre résolution, à savoir uniquement les matrices D_0 et D_2. Nous instancions également la matrice de masse M. Nous avons découpé la grille spatiale régulièrement, mais il est tout à fait envisageable de générer des éléments de taille quelconque car les éléments finis Lagrangiens n'imposent pas un maillage uniforme. Pour la suite des calculs, nous supposerons utiliser un maillage uniforme.

Le vecteur de conditions initiales est déterminé sans grande difficulté grâce à la solution analytique implémentée dans le script "u_anal.m". Le vecteur u construit, ainsi que la discrétisation spatiale implémentée juste après seront donnés à l'intégrateur temporel. Nous préparons également les paramètres d'intégration numérique qui vont être utilisés pour le traitement du terme source. Nous posons $nquad = 2$ qui calculera 3 abscisses $xquad$ avec ses poids $wquad$ respectifs. Modifier ce paramètre n'influence pas la qualité de la solution, nous le laisserons dès lors fixé à 2.

Enfin, après être rentré dans le sous-programme de calcul de l'EDP, détaillé dans la section suivante, nous affichons le graphique de la solution numérique en vis-à-vis avec la solution analytique ainsi que le graphe des erreurs sur le domaine

3.2.2 Sous-programme de calcul du système d'EDO : BurgerFisherLagr.m

"BurgerFisherLagr.m" va reconstituer le membre de droite de notre EDP à résoudre. Les deux termes linéaires u et u_{xx} sont très facilement accessibles grâce aux matrices de dérivation précédemment définies. La majeure subtilité en terme de programmation réside dans le calcul numérique du terme source et dans l'assemblage élément par élément de ce terme source dans le système d'EDO.

Nous avons séparé le terme source en deux : $src(u) = src_1(u) + src_2(u) = c_1(t)u^2 u_x + c_2(t)u^4$. En se souvenant qu'un élément affecte 2 lignes dans le vecteur $\overline{f}_{NL}(\overline{u})$ complet, nous avons également séparé le traitement de la première composante de la deuxième composante du vecteur de dimension 2 que génère un élément pour le terme source. C'est ce qui est fait lors des calculs des valeurs de sortie de integrand1_1 et integrand2_1 pour $src_1(u)$. Il faut enfin appliquer la formule de Gauss pour l'intégration numérique en faisant la somme pondérée par les poids $wquad$ aux abscisses $xquad$ déterminées dans le programme principal via l'instruction "feval". Le vecteur $yu2ux\_1$, de dimension $n-1$, contribue au vecteur complet $\overline{f}_{NL}(\overline{u})$ de la première à l'avant-dernière ligne (d'où l'instruction $u_t(1:n-1)$) et le vecteur $yu2ux\_2$ de la deuxième à la dernière ligne, comme le suggère l'assemblage expliqué dans les rappels théoriques.

Un processus analogue pour la deuxième partie du terme source peut être appliqué. Il fait intervenir les scripts integrand1_2 et integrand2_2.

Le système d'EDO sera complet après l'ajout des CL sur la première ($u_t(1)$) et dernière ligne ($u_t(n)$) du système.

3.2.3 Traitement du terme source : integrand 1_1, 2_1, 1_2, 2_2

Le premier indice des fonctions "integrand" indique quelle est la composante traitée du vecteur de dimension 2 représentant le terme source. Le deuxième indice montre si le calcul s'effectue pour $src_1(u) = c_1(t)u^2u_x$ ou $src_2(u) = c_2(t)u^4$. Ces 4 scripts implémentent simplement le terme source évalué en $\hat{u}$, c'est-à-dire en remplaçant u par son modèle mathématique lagrangien pour un élément.

L'implémentation des "integrand" relatifs à src_2 ne demande pas de précaution particulière, il faut juste ne pas oublier le facteur $\frac{h}{2}$ qui multiplie l'ensemble du vecteur, comme montré dans les rappels théoriques. En revanche, les "integrand" concernant le terme en u^2u_x est particulier. Étant donné que nous travaillons en coordonnées réduites, le changement de coordonnées $\frac{\partial u}{\partial x} = \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x}$ va introduire un facteur $\frac{\partial \xi}{\partial x} = \frac{2}{h}$ qui va se simplifier avec le coefficient $\frac{h}{2}$ présent quoi qu'il arrive dans l'expression de $\overline{f}_{NL}(\overline{u})$. Les integrand1_1 et integrand2_1 n'ont donc pas de facteur $h/2$ qui multipliera la sortie de ces programmes.

3.3 Résultats numériques

Lançons le simulateur avec les paramètres quelconques dans un premier temps. Les graphiques ci-dessous vont montrer la solution numérique et ses erreurs dans les conditions suivantes : ode23, 250 points, tolérances 10^{-5}.

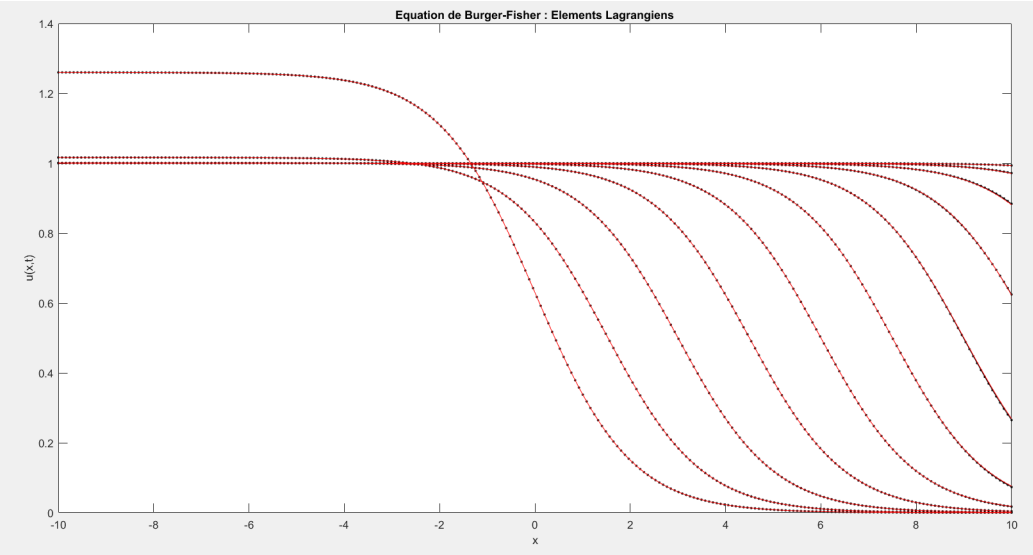


FIGURE 7: Solution numérique VS Solution analytique : Éléments Lagrangiens

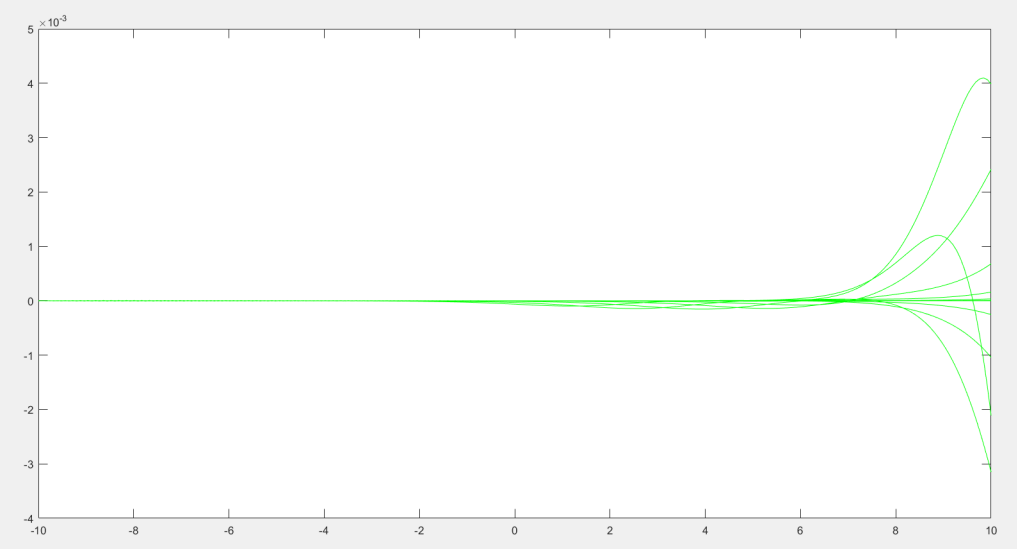


FIGURE 8: Erreurs pour chaque instant de visualisation : Éléments Lagrangiens

La solution numérique coïncide avec la solution analytique. Nous allons maintenant faire varier les 3 paramètres et en voir les conséquences sur les erreurs et le temps d'exécution.

Tolérances = 10^-5		Nombre de points									
Intégrateurs		50		100		250		500		750	
		temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err
non-stiff	ode45	0,41	0,0115	1,21	0,0068	8,8	0,0038	51	0,0026	152	0,0021
	ode23	0,31	0,0115	0,82	0,0068	5,5	0,0038	31,6	0,0026	94	0,0021
	ode113	0,4	0,0115	0,99	0,0068	6,5	0,0038	36,4	0,0026	109	0,0021
stiff	ode15s	0,36	0,0115	0,39	0,0068	0,53	0,0038	0,85	0,0026	1,31	0,0021
	ode23s	3,36	0,0116	8,3	0,0068	33	0,0038	117	0,0026	259	0,0021
	ode23tb	0,46	0,0115	0,58	0,0068	1,05	0,0038	2,0	0,0026	3,93	0,0021
	ode23t	0,39	0,0115	0,44	0,0067	0,64	0,0037	1,02	0,0025	1,7	0,0021

FIGURE 9: Comparaisons intégrateurs VS nombre de points

Tout d'abord, analysons les résultats d'une variation de points entre 50 et 750 avec une tolérance fixée à 10^{-5}. Tout d'abord, pour les intégrateurs non stiff, on peut remarquer que le temps de calcul augmente de manière considérable avec le nombre de points. En effet, alors qu'à 50 points le temps était aux alentours des 0.3 s, il grimpe jusqu'à 152 s à 750 points. L'erreur quant à elle ne diminue pas de manière significative (0.0115 à 50 points et 0.0021 à 750 points). Pour ce qui est des intégrateurs stiff, ode15s voit son temps de calcul augmenter de manière modérée avec le nombre de points. Pour ce qui est de ode23tb et ode23t, la tendance est plus prononcée. Ode23s quant à lui présente un temps de calcul bien supérieur aux autres intégrateurs et ce, à partir de 50 points, il n'est donc pas judicieux de choisir celui-ci.
Globalement, on peut dire qu'il est intéressant de conserver un nombre de points relativement peu élevé (250) couplés aux intégrateurs qui préservent un temps raisonnable (ode15s et ode23t) pour avoir une erreur décente. Après analyse, on observe que l'intégrateur non stiff le plus rapide pour 250 points est ode23.

Fixons maintenant le nombre de points à 250 et analysons l'influence d'une tolérance variant de 10^{-3} à 10^{-7}.

Nombre de points = 250		Tolérances									
Intégrateurs		10^-3		10^-4		10^-5		10^-6		10^-7	
		temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err
non-stiff	ode45	8,7	0,0045	8,8	0,0038	8,8	0,0038	8,8	0,0038	8,8	0,0038
	ode23	5,5	0,0083	5,5	0,0038	5,5	0,0038	5,5	0,0038	5,5	0,0038
	ode113	6,3	0,0072	6,4	0,004	6,6	0,0038	6,4	0,0038	6,5	0,0038
stiff	ode15s	0,45	0,0074	0,48	0,0038	0,52	0,0038	0,56	0,0038	0,63	0,0038
	ode23s	6,6	0,0099	15	0,0043	33	0,0038	73	0,0038	161	0,0038
	ode23tb	0,46	0,0038	0,66	0,0037	1	0,0038	1,9	0,0038	3,6	0,0038
	ode23t	0,44	0,0072	0,51	0,0039	0,63	0,0037	0,75	0,0038	1,1	0,0038

FIGURE 10: Comparaisons intégrateurs VS tolérances à 250 points

Tout d'abord, on remarque que pour les intégrateurs non stiff, imposer une tolérance plus sévère n'influence pratiquement pas le temps de calcul et diminue légèrement l'erreur (en 10^{-3} on était à environ 0.007 et en 10^{-7} à 0.0038). Ces derniers stagnent à une tolérance élevée. Pour ce qui est des intégrateurs stiff, le temps de calcul augmente plus ou moins fort selon l'intégrateur lorsque la tolérance diminue et les erreurs convergent très rapidement vers une valeur de 0.0038, la même valeur que l'erreur des intégrateurs non stiff. Diminuer encore plus la tolérance (jusque 10^{-10}) n'apporte rien en précision et augmente considérablement le temps de calcul pour les intégrateurs stiff. Le choix de l'intégrateur se basera donc sur une tolérance de 10^{-7} en choisissant ode15s.

Si nous regardons de plus près les graphiques des erreurs pour le simulateur lagrangien en annexe, les erreurs sont concentrées à l'extrémité droite du domaine. Les graphiques sont très ressemblants les uns par rapport aux autres à cause de cette valeur de 0.0038 pour *ec* qui semble être un point de convergence d'erreur (qui dépend du nombre de points).

3.4 Conclusions intermédiaires

Nous pouvons conclure, pour le simulateur Lagrangien :

— Les intégrateurs sont très sensibles au nombre croissant de points ;

— Pour un même nombre de points, les intégrateurs sont indifférents en ce qui concerne l'erreur, mais se distinguent par leur temps d'exécution ;

— L'erreur converge vers une valeur lorsque les tolérances diminuent. Cette valeur est fonction du nombre de points uniquement ;

— Les erreurs sont localisées au bord droit du domaine

Le simulateur Lagrangien idéal serait, pour 250 points :

— Nombre de points : 250

— Intégrateur : ode15s

— Tolérances : 10^{-7}

4 Collocation Orthogonale

4.1 Rappel théorique

Repartons des résidus pondérés exprimés de manière générale à la section précédente :

$$\int_{\Omega(x)} W_l F\left(\sum_{k=1}^{E} u_{kt}(t) N_k(x)\right) dx = \alpha_0 \int_{\Omega(x)} W_l \hat{u} dx + \alpha_1 \int_{\Omega(x)} W_l \hat{u}_x dx + \alpha_2 \int_{\Omega(x)} W_l \hat{u}_{xx} dx + \int_{\Omega(x)} W_l src(\hat{u}) dx$$

Résidus pondérés sur le domaine Ω

Cette fois-ci, les fonctions de pondération W_l ne sont pas les fonctions de base comme le suggère la méthode de Galerkin mais des impulsions de Dirac en des points de collocation choisis de manière judicieuse. De plus, les fonctions de base N_k sont les éléments hermitiens. Un élément quelconque est donc exprimé, en coordonnée réduite, par :

$$\hat{u}(\xi,t) = u_a(t)N_1(\xi) + u_{a\xi}(t)N_2(\xi) + u_b(t)N_3(\xi) + u_{b\xi}(t)N_4(\xi)$$

Si nous avons N éléments, nous aurons 2N+2 inconnues. En choisissant 2 points de collocation par élément, nous aurons 2N relations. Les deux relations restantes seront complétées par les conditions aux limites qui nous sont fournies par la solution analytique. Pour 2 points de collocation, les emplacements optimaux en terme de précision sont en $\xi_1 = -\frac{1}{\sqrt{3}}$ et $\xi_2 = \frac{1}{\sqrt{3}}$.

En remplaçant les fonctions de pondération par les impulsions de Dirac en x_{col}, les résidus pondérés sur un élément hermitien s'établissent comme ceci :

$$\int_{\Omega_k} \delta(x_{col})\hat{u}_t dx = \alpha_0 \int_{\Omega_k} \delta(x_{col})\hat{u} dx + \alpha_1 \int_{\Omega_k} \delta(x_{col})\hat{u}_x dx + \alpha_2 \int_{\Omega_k} \delta(x_{col})\hat{u}_{xx} dx + \int_{\Omega_k} \delta(x_{col})src(\hat{u})dx$$

$$\hat{u}_t(x_{col}) = \alpha_0\hat{u}(x_{col}) + \alpha_1\hat{u}_x(x_{col}) + \alpha_2\hat{u}_{xx}(x_{col}) + src(\hat{u}(x_{col}))$$

À nouveau, cette expression est celle d'un système d'EDO qui sera solutionné par les intégrateurs temporels de Matlab. Remarquons également que les intégrales du terme source qui ont fait l'objet d'une intégration numérique dans le simulateur via les éléments Lagrangiens n'ont plus raison d'être grâce à la collocation orthogonale.

En exprimant, par élément, chaque terme de la dernière relation, nous remarquons que l'élément intervient dans deux lignes du système et fait intervenir 4 inconnues. La "matrice-pavé" est de dimension 2x4. Les deux relations restantes représentant les conditions aux limites pour fermer le système seront placées en première et dernière position sous forme d'équations algébriques. Il est donc possible de retrouver la matrice de masse M et les matrices de dérivations spatiales D_0, D_1, D_2 de la collocation orthogonale après assemblage. Ces matrices sont construites et fournies dans les scripts Matlab :

$$M\overline{u_t} = \alpha_0 D_0\overline{u} + \alpha_1 D_1\overline{u} + \alpha_2 D_2\overline{u} + src(\hat{u})$$

Par élément, le terme source intervient également dans deux lignes du système et se construit comme ceci :

$$\begin{pmatrix} 0 \\ src(u_{12}(\xi_1)) \\ src(u_{12}(\xi_2)) \\ src(u_{23}(\xi_1)) \\ src(u_{23}(\xi_2)) \\ \vdots \\ src(u_{N-1N}(\xi_1)) \\ src(u_{N-1N}(\xi_2)) \\ 0 \end{pmatrix}$$

Il faut enfin placer les conditions aux limites sur la première et dernière ligne du système d'EDO. Nous avons des conditions aux limites de type Dirichlet. En vertu de l'expression des résidus pondérés pour les bords du domaine, il faut cette fois-ci placer $+(u_1 - u(x_0,t))$ et $+(u_N - u(x_L,t))$ contrairement aux signes négatifs présents dans les CL des éléments Lagrangiens.

4.2 Implémentation

Rappelons l'EDP à résoudre :

$$\frac{\partial u}{\partial t} = \frac{2}{[1+e^{-6t}]^{\frac{2}{3}}} u^2 \frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{2}{1+e^{-6t}} u(1-u^3)$$

Nous gardons toujours la même discrétisation temporelle, les mêmes CI et CL que les deux simulateurs précédents. Sous la même structure à 3 programmes Matlab que pour le simulateur Lagrangien, voici comment nous avons conçu le simulateur en utilisant la collocation orthogonale (voir annexes pour les détails des codes)

4.2.1 Programme principal : Colloc.m

Nous définissons d'abord les variables du problème pour ensuite générer la grille spatiale en fonction du nombre de points n (il y a donc $2n$ inconnues). Nous avons décidé de découper le domaine de manière régulière mais en toute généralité, la collocation orthogonale peut s'appliquer sur des éléments de taille quelconque.

Étant donné que nous travaillons avec des éléments hermitiens, le vecteur de conditions initiales à fournir à l'intégrateur temporel est le vecteur :

$$\begin{pmatrix} x_1 \\ x_{1\xi} \\ x_2 \\ x_{2\xi} \\ \vdots \\ \vdots \\ x_{N-1} \\ x_{N-1\xi} \\ x_N \\ x_{N\xi} \end{pmatrix}_0 = \begin{pmatrix} u_1 \\ (h_1/2)u_{1x} \\ u_2 \\ ((h_1+h_2)/4)u_{2x} \\ \vdots \\ \vdots \\ u_{N-1} \\ ((h_{N-2}+h_{N-1})/4)u_{N-1x} \\ u_N \\ (h_{N-1}/2)u_{Nx} \end{pmatrix}$$

Le vecteur hCI permet la construction de ce vecteur initial.

Après avoir instancié les matrices de masse et de dérivations, nous pouvons entrer dans le sous programme de calcul de l'EDP avec les instants de visualisation et le vecteur initial, reconstitué en alternant les CI sur u et les CI sur u_ξ grâce aux conditions initiales fournies respectivement par la solution analytique et la dérivée de la solution analytique (implémentés dans "$u\_anal.m$" et "$u\_anal\_der.m$"). La sortie de ce sous-programme contient la solution numérique, que nous afficherons sur un graphique, superposé à la solution analytique. Nous calculons enfin l'erreur entre les deux solutions, en le nuançant avec le graphe des erreurs sur le domaine entier pour chaque instant de visualisation.

4.2.2 Sous-programme de calcul du système d'EDO : BurgerFisherColloc.m

Cette partie de l'implémentation sert à construire le système d'EDO détaillé dans le rappel théorique que l'intégrateur de Matlab va résoudre itérativement.

Notre équation contient deux termes linéaires et deux termes non linéaires. Dans les codes, on distingue la partie linéaire, facilement assemblée à u_t via les matrices de dérivation, de la partie non linéaire qui est évaluée via les deux programmes qui vont traiter le terme source.

Après avoir placé les termes linéaires, il faut placer les termes non linéaires, sorties des sous-programmes integrant1 et integrant2, de manière minutieuse. Si l'on reprend l'expression du vecteur correspondant au terme source dans le rappel théorique, le premier point de collocation contribue aux lignes en commençant par la deuxième ligne et termine à l'antépénultième ligne (d'où l'assemblage du terme source sur $u_t(2:2:2*n-2)$) et le deuxième point de collocation contribue de la troisième ligne à l'avant-dernière ligne, également par pas de 2 (assemblage sur $u_t(3:2:2*n-1)$).

Enfin, l'implémentation des conditions aux limites se fait de manière algébrique sur les première et dernière ligne du système. Les instructions s'appliquent donc bien aux lignes $u_t(1)$ et $u_t(2*n)$ mais font intervenir respectivement les inconnues $u(1)$ et $u(2*n-1)$ puisque rappelons-le, le vecteur $\overline{u}$ contient alternativement u et u_ξ.

4.2.3 Traitement du terme source : integrand1.m + integrand2.m

Le terme source est :

$$src(u) = \frac{2}{[1+e^{-6t}]^{\frac{2}{3}}}u^2\frac{\partial u}{\partial x} - \frac{2}{1+e^{-6t}}u^4 = src_1(u) + src_2(u)$$

Pour plus de clarté dans l'implémentation, nous avons décidé de traiter séparément les termes sources. "integrand1.m" traite le terme en u^2u_x et "integrand2.m" le terme en u^4. Le contenu de ces deux scripts Matlab n'est que l'évaluation du terme source correspondant quand u est remplacé par son modèle mathématique $\hat{u}$. Il faut prendre garde à un dernier détail qui est l'utilisation des coordonnées réduites qui introduit un facteur $2/h$ par ordre de dérivation présent dans le terme source. En effet, le changement de coordonnées multiplie la contribution par $\frac{\partial \xi}{\partial x} = \frac{2}{h}$. "integrand1.m" voit donc sa sortie multipliée par $\frac{2}{h}$.

4.3 Résultats numériques

Les intégrateurs utilisés dans ces tests sont ode23t et ode15s étant donné qu'ils sont les deux seuls capables de résoudre les systèmes différentiels-algébriques. Nous avons affaire à un système différentiel algébrique à cause de l'implémentation particulière des conditions aux limites. Une implémentation de base avec ode15s, tolérances 10^{-5} et 250 points est montrée ci-contre :

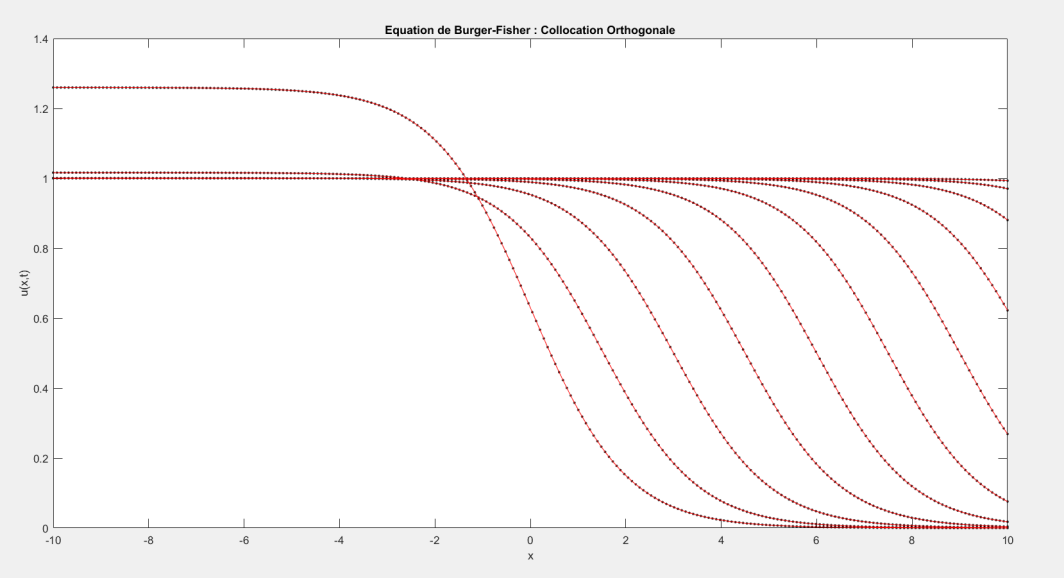


FIGURE 11: Solution numérique VS Solution analytique : Collocation orthogonale

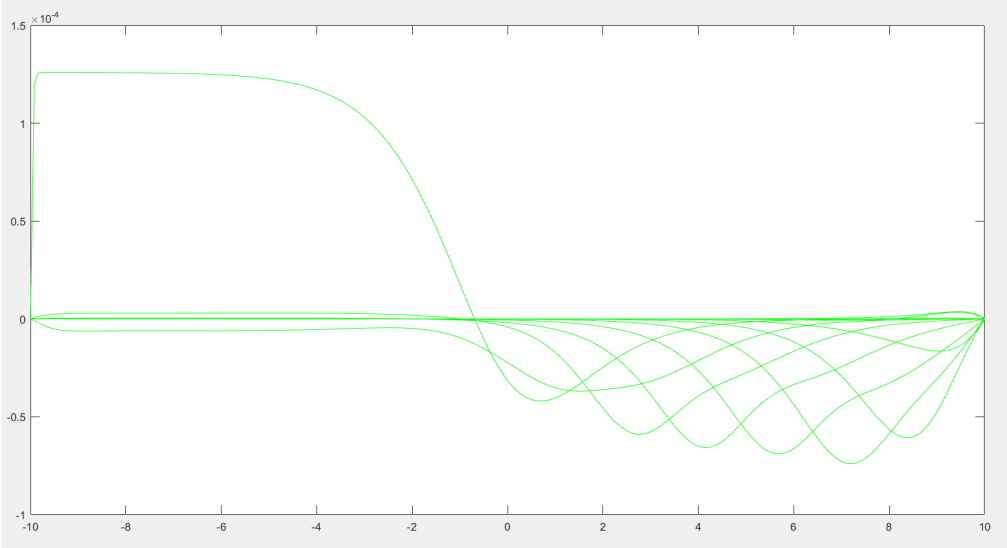


FIGURE 12: Erreurs pour chaque instant de visualisation : Collocation orthogonale

À nouveau, la solution numérique se superpose parfaitement à la solution analytique sur ce graphique. Parcourons en détail l'influence des paramètres que nous pouvons modifier sur le résultat final en terme de performances et d'erreur de calcul.

Ci-dessous sont représentés les résultats pour 250 points en fonction de l'intégrateur et des tolérances :

Nombre de points = 250	Tolérances									
Intégrateurs	10^-3		10^-4		10^-5		10^-6		10^-7	
	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err
ode15s	0,9	0,0058	0,79	7,85E-04	0,91	3,0110E-04	1,0	2,5062E-04	1,2	2,5033E-04
ode23t	0,7	0,0071	0,83	0,0017	1,1	5,5429E-04	1,3	3,0162E-04	1,8	2,6190E-04

FIGURE 13: Comparaisons intégrateurs VS tolérances : 250 points

Si nous regardons l'influence de la tolérance sur l'erreur et le temps lorsque le nombre de points est fixé à 250, on remarque que la tolérance influence le temps de calcul de manière plutôt significative (entre 0.7 et 1.8 s). Pour ce qui est de l'erreur, elle est d'environ 6.10^{-3} et diminue ensuite jusqu'à $2,5.10^{-4}$ à 10^{-7} de tolérance. L'objectif est donc de trouver une tolérance qui diminuerait l'erreur sans pour autant faire trop grimper notre temps de calcul. En analysant le tableau des résultats, on peut atteindre 10^{-4} en erreur tout en gardant un temps de calcul raisonnable. En effet, en utilisant ode15s avec une tolérance de 10^{-5}, on optimise nos résultats c'est donc avec la tolérance 10^{-5} que nous avons travaillé pour les tests à tolérance fixe.

Regardons maintenant les résultats collectés lorsque la tolérance est fixée à 10^{-5} et que le nombre de points varie entre 50 et 750.

Tolérances = 10^-5	Nombre de points									
Intégrateurs	50		100		250		500		750	
	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err
ode15s	0,36	1,0966E-04	0,40	1,6985E-04	0,91	3,0110E-04	2,6	4,3515E-04	6,8	5,2609E-04
ode23t	0,38	2,5223E-04	0,41	3,0705E-04	1,1	5,5429E-04	3,0	6,6401E-04	8,5	9,5106E-04

FIGURE 14: Comparaisons intégrateurs VS nombre de points

Globalement, le temps de calcul est compris entre 0.36 et 8.5s. Les erreurs quant à elles varient en augmentant entre 1.10^{-4} et 9.10^{-4}. Cela semble curieux puisque nous rendons plus fin le maillage spatial mais la diminution des erreurs de troncature est inhibée par les erreurs d'arrondi que nous cumulons avec un système d'EDO plus grand. Étant donné que notre objectif est de minimiser le temps de calcul et l'erreur, le choix le plus judicieux serait de prendre un nombre de points égal à 50 avec ode15s. Malheureusement, un maillage si grossier n'est pas satisfaisant non plus. Moyennant un temps de calcul plus grand, nous opterons pour un nombre de points égal à 250.

Concernant les graphiques d'erreurs pour la collocation orthogonale, également présentés en annexe, nous pouvons remarquer que ces courbes se stabilisent vers une allure particulière avec une première courbe située à gauche et les autres courbes à droite dans le domaine spatial. Il est donc possible de conclure qu'il faut un certain seuil de tolérance avant de converger vers l'allure correcte des erreurs (dès 10^{-6} pour ode15s, à 10^{-7} pour ode23t).

Comme en différences finies, la majorité des erreurs apparaissent à l'endroit où les courbes analytiques commencent à descendre, aux environs de $x = 0$.

4.4 Conclusions intermédiaires

Globalement, nous pouvons conclure que :

— Étant donné la configuration particulière des conditions aux limites, nous avons affaire à un système différentiel-algébrique. Les intégrateurs utilisés sont donc ode15s et ode23t ;

— Imposer une tolérance sévère augmente le temps de calcul de manière non négligeable ;

— L'erreur diminue lorsque la tolérance diminue ;

— Le temps de calcul augmente considérablement avec le nombre de points.

En conclusion, il s'agira de trouver la tolérance, le nombre de points et l'intégrateur qui minimisent l'erreur et le temps de calcul. Nous retenons qu'il faudra choisir un nombre de points relativement petit (tout en gardant une solution numérique graphiquement lisse) et une tolérance raisonnable par rapport au temps de calcul qu'elle engendrera. Pour ce qui est de l'intégrateur, ode15s est meilleur que ode23t dans tous les scénarios possibles de l'utilisation de ce simulateur.

Notre simulateur via la collocation orthogonale serait :

— Nombre de points : 250 points
— Intégrateur : ode15s
— Tolérances : 10^{-5}

5 Comparaison des 3 simulateurs

L'intégrateur utilisant les différences finies nous permet d'obtenir des résultats dont l'erreur est de l'ordre de 10^{-7} en un temps très court (autour des 0.3s sur notre machine). Ceci est le meilleur résultat que nous avons pu obtenir lors de nos différents tests. Notre simulateur via les éléments Lagrangiens nous fournis dans les meilleures configurations une erreur de l'ordre de 10^{-3} pour des temps de calculs dépassant la seconde. En faisant varier le nombre de subdivisions de notre espace, nous pouvons améliorer soit le temps de calcul au détriment de la précision du résultat, soit la précision au détriment du temps de calcul. Quant à notre simulateur basé sur la collocation orthogonale, nous obtenons des erreurs de l'ordre de 10^{-4} pour un temps de calcul de 0,4 secondes dans la meilleure configuration. Ces résultats sont bien moins intéressant que ceux obtenus grâce à la méthode des différences finies.

Une autre comparaison peut être faite. Le paragraphe précédent considérait un même nombre de points pour les 3 simulateurs. Si nous voulons mettre les simulateurs sur le même pied d'égalité concernant la taille du système d'EDO à résoudre, il faudrait considérer la collocation orthogonale à 125 points tandis que les différences finies et éléments lagrangiens resteraient à 250 points. En effet, l'utilisation théorique des éléments hermitiens dans le cadre de la collocation orthogonale génère un système de taille $2 * n$ (avec n le nombre de points). En prenant respectivement 250, 250 et 125 points pour les différences finies, éléments lagrangiens et collocation, les résultats obtenus concerneraient la même taille de problème.

Ci-dessous sont affichés, pour rappel, les tableaux des tolérances en fonction des intégrateurs à 250 points pour les deux premiers simulateurs, mais également le même tableau à 125 points pour la collocation orthogonale :

	nombre de points = 250 ; D1=five_point_biased_upwind_D1(x,1) ; D2=five_point_centered_D2(x) ; Jpattern = non							
	Tolérances	10^-3		10^-5		10^-7		10^-9	
	Intégrateurs	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err
non-stiff	ode45	0,38	0,0019	0,38	2,11E-05	0,38	2,02E-07	0,58	1,77E-07
	ode23	0,30	0,0032	0,29	3,07E-05	0,29	4,07E-07	0,51	1,78E-07
	ode113	0,48	0,0092	0,45	5,44E-06	0,46	4,84E-07	0,48	1,77E-07
stiff	ode15s	0,26	0,0067	0,27	6,60E-05	0,32	4,97E-07	0,35	1,72E-07
	ode23s	0,78	0,0084	3,8	3,03E-04	19	1,17E-05	90,6	5,34E-07
	ode23tb	0,23	0,0061	0,29	2,50E-04	0,60	1,17E-05	1,95	6,06E-07
	ode23t	0,25	0,0064	0,27	2,72E-04	0,43	1,61E-05	1,1	8,44E-07

Différences finies à 250 points

Nombre de points = 250		Tolérances									
Intégrateurs		10^-3		10^-4		10^-5		10^-6		10^-7	
		temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err
non-stiff	ode45	8,7	0,0045	8,8	0,0038	8,8	0,0038	8,8	0,0038	8,8	0,0038
	ode23	5,5	0,0083	5,5	0,0038	5,5	0,0038	5,5	0,0038	5,5	0,0038
	ode113	6,3	0,0072	6,4	0,004	6,6	0,0038	6,4	0,0038	6,5	0,0038
stiff	ode15s	0,45	0,0074	0,48	0,0038	0,52	0,0038	0,56	0,0038	0,63	0,0038
	ode23s	6,6	0,0099	15	0,0043	33	0,0038	73	0,0038	161	0,0038
	ode23tb	0,46	0,0038	0,66	0,0037	1	0,0038	1,9	0,0038	3,6	0,0038
	ode23t	0,44	0,0072	0,51	0,0039	0,63	0,0037	0,75	0,0038	1,1	0,0038

Éléments lagrangiens à 250 points

Nombre de points = 125	Tolérances									
Intégrateurs	10^-3		10^-4		10^-5		10^-6		10^-7	
	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err	temps (s)	err
ode15s	0,44	0,0034	0,39	5,12E-04	0,43	1,9498E-04	0,45	1,7671E-04	0,53	1,7664E-04
ode23t	0,35	0,0052	0,39	0,0013	0,48	4,0232E-04	0,55	2,1705E-04	0,55	2,1705E-04

Collocation orthogonale à 125 points

Comme expliqué plus haut, le simulateur par différences finies fournit les meilleurs résultats en terme de précision avec des tolérances très faibles. Seules les configurations optimales permettent d'atteindre ces résultats extraordinaires. Cet avantage ne s'obtient qu'au prix d'une réflexion autour du choix des schémas de différences finies. Le mauvais schéma de dérivée seconde par exemple empêche toute amélioration de l'erreur par d'autres moyens. L'option JPattern implémentée permet de réduire drastiquement le temps de calcul pour les intégrateurs stiff.

Le simulateur basé sur les éléments Lagrangiens sera, lui, plus facilement implémentable. En effet, les éléments Lagrangiens n'intervenant que dans deux lignes du système d'EDO à résoudre, les calculs de la matrice de masse ainsi que de celles d'ordre 0, 1 et 2 deviennent simples à implémenter. De plus, la variation des différents paramètres du simulateur ne change presque pas le résultat obtenu, il n'est donc pas nécessaire de tester plusieurs possibilités pour trouver la configuration parfaite. La contrepartie est que les résultats obtenus via ce simulateur ne sont pas satisfaisants comparativement aux deux autres.

Le simulateur via la collocation orthogonale est utile lorsque les tolérances exigées sont élevées, car le temps de calcul n'augmente que très peu lorsque les tolérances diminuent. Les erreurs sont également très faibles pour le peu de points utilisés. Malheureusement, l'implémentation est particulière et requiert beaucoup de manipulations, avec entre autres les conditions aux limites transformant le système à résoudre en un système différentiel-algébrique (limitant le nombre d'intégrateurs utilisables) et l'utilisation des éléments hermitiens qui rend la reconstitution du système d'EDO complexe.

6 Annexes

Voici la copie des codes implémentés pour concevoir les 3 simulateurs :

6.1 Codes des différences finies

DiffFinies.m

```
close all
clear all
%...
%Debut du chronometre
%...
tic

%...
%Variables globales du probleme
%...
global x0 xL n D1 D2;

%...
%Grille spatiale
%...
x0=-10.0;
xL=10.0;
n=501;
dx=(xL-x0)/(n-1);
x=[x0:dx:xL]';

%...
%Conditions initiales
%...
u=zeros(n,1);
for i=1:n
    u(i) = u_anal(x(i),0);
end;

%...
%Schemas de differences finies pour les derivees spatiales
%...

%Derivee premiere
%D1=five_point_biased_upwind_D1(x,1);
%D1=five_point_biased_upwind_D1(x,-1);
%D1=five_point_centered_D1(x);

%D1=four_point_biased_upwind_D1(x,1);
%D1=four_point_biased_upwind_D1(x,-1);
%D1=four_point_upwind_D1(x,1);
%D1=four_point_upwind_D1(x,-1);
```

```
%D1=three_point_centered_D1(x);
%D1=three_point_upwind_D1(x,1);
%D1=three_point_upwind_D1(x,-1);

%D1=two_point_upwind_D1(x,1);
D1=two_point_upwind_D1(x,-1);

%Derivee seconde
D2=five_point_centered_D2(x);
%D2=three_point_centered_D2(x);

%...
%Matrice pour le JPattern
%...
J = [spones(eye(n)) + spones(D1) + spones(D2)];
J = spones(J);
J = sparse(J);

%...
%Parametres de l'integration temporelle
%...
t=[0:0.5:5];
%options = odeset('RelTol',1e-5,'AbsTol',1e-5,'stats','on');
options = odeset('RelTol',1e-5,'AbsTol',1e-5,'stats','on','JPattern',J);

%...
%Integration temporelle
%...

%Solveurs non stiff
%[tout, yout] = ode45(@BurgerFisherDiff,t,u,options);
%[tout, yout] = ode23(@BurgerFisherDiff,t,u,options);
%[tout, yout] = ode113(@BurgerFisherDiff,t,u,options);

%Solveurs stiff
%[tout, yout] = ode15s(@BurgerFisherDiff,t,u,options);
[tout, yout] = ode23s(@BurgerFisherDiff,t,u,options);
%[tout, yout] = ode23tb(@BurgerFisherDiff,t,u,options);
%[tout, yout] = ode23t(@BurgerFisherDiff,t,u,options);

%...
%Rapatriement des conditions aux limites
%...
for k=1:length(tout)
    yout(k,1) = u_anal(x0,tout(k));
    yout(k,n) = u_anal(xL,tout(k));
end
```

```
%...
%Fin du chronometre
%...
tcpu = toc

%...
%Affichage de la solution analytique VS solution numerique
%...

%Solution numerique
plot(x,yout,'.-k');
xlabel('x');
ylabel('u(x,t)');
title('Equation de Burger-Fisher : Differences Finies')
hold on

%Solution analytique
for k=1:length(tout)
    for i=1:n
        yexact(k,i) = u_anal(x(i),tout(k));
    end;
    plot(x,yexact(k,:),'r')
    hold on
end;

%...
%Calcul de l'erreur + Graphe de l'erreur
%...
figure;
e=ec(yexact, yout,x)
```

BurgerFisherDiff.m

```
function ut = BurgerFisherDiff(t,u)
%...
%Variables globales
%...
global x0 xL n D1 D2;
t

%...
%CL a gauche
%...
u(1) = u_anal(x0,t);

%...
%CL a droite
%...
u(n) = u_anal(xL,t);
```

```
%...
%Construction de ut = a.u2.ux + uxx + b(u-u4)
%...

%Terme : uxx
uxx = D2*u;

%Terme : a.u2.ux
ux = D1*u;
u2=u.*u;
a=2/((1+exp(-6*t))^(2/3));
fx = ux.*u2;
fx=a*fx;

%Terme : b(u-u4)
b=2/(1+exp(-6*t));
u3=u2.*u;
f=-u.*(u3-1);
f=f*b;

%Assemblage de ut
ut = f+fx+uxx;

%...
%Fixation des conditions aux limites
%...
ut(1) = 0;
ut(n) = 0;
```

ec.m

```
function e = ec(yanal, ynum,x)

diff=yanal-ynum;
for i=1:11
    a(i)=norm(diff(i,:));
    plot(x,diff(i,:),'g');
    hold on
end
e=sum(a);
e=e/11;

end
```

u_anal.m

```
function u=u_anal(x,t)
    u=0.5*(1+exp(-6*t))^(1/3)*(1-tanh((x-3*t)/2));
end
```

6.2 Codes des éléments Lagrangiens

Lagrange.m

```
close all
clear all
%...
%Debut du chronometre
%...
tic

%...
%Variables globales du probleme
%...
global wquad xquad
global n dx x0 xL D2 h D0

%...
%Grille spatiale
%...
x0 = -10;
xL = 10;
n = 101;
nel = n-1;
dx = (xL - x0)/(n-1);
x = (x0:dx:xL)';
h = (x(2:n) - x(1:n-1))';
ne = 1;

%...
%Matrices de differentiation pour elements Lagrangiens
%...
D0 = lagrD0_1(h(1),n);
D2 = lagrD2_1(h(1),n);
M = masseL1(h(1),n,ne);

%...
%Conditions initiales
%...
u = zeros(1,n);
for i = 1:n
    u(i) = u_anal(x(i),0);
end

%...
%Instants de visualisation
%...
dt = 0.5;
t = (0:dt:5);
nt = length(t);

%...
```

```
%Parametres d'integration numerique
%...
nquad = 2;
beta = .5./sqrt(1-(2*(1:nquad)).^(-2));
T = diag(beta,1) +diag(beta,-1);
[V,D] = eig(T);
xquad = diag(D);
[xquad,i] = sort(xquad);
wquad = 2*V(1,i).^2;

%...
%Integration temporelle
%...
options = odeset('Mass',M,'RelTol',1e-5,'AbsTol',1e-5,'stats','on');

%Solveurs non stiff
%[tout, yout] = ode45(@BurgerFisherLagr,t,u,options);
%[tout, yout] = ode23(@BurgerFisherLagr,t,u,options);
%[tout, yout] = ode113(@BurgerFisherLagr,t,u,options);

%Solveurs stiff
[tout, yout] = ode15s(@BurgerFisherLagr,t,u,options);
%[tout, yout] = ode23s(@BurgerFisherLagr,t,u,options);
%[tout, yout] = ode23tb(@BurgerFisherLagr,t,u,options);
%[tout, yout] = ode23t(@BurgerFisherLagr,t,u,options);

%...
%Fin du chronometre
%...
tcpu = toc

%...
%Affichage solution analytique VS solution numerique
%...

%Solution numerique
plot(x,yout,'.-k')
xlabel('x');
ylabel('u(x,t)');
title('Equation de Burger-Fisher : Elements Lagrangiens')
hold on

%Solution analytique
for k = 1:length(tout)
    for i = 1:n
        yexact(k,i) = u_anal(x(i),tout(k));
    end
    plot(x,yexact(k,:),'r')
    hold on
```

```
end

%...
%Calcul de l'erreur + Graphe de l'erreur
%...
figure;
e=ec(yexact, yout,x)
```

BurgerFisherLagr.m

```
function ut = BurgerFisherLagr(t,u)
%...
%Variables globales
%...
global wquad xquad
global n x0 xL D2 u1 D0 h
u1 = u;
t

c1 = 2/((1+exp(-6*t))^(2/3));
c2 = (-2)/(1+exp(-6*t));

%...
%Construction de ut = -c2.D0.u + D2.u + c1.src1(u) + c2.src2(u)
%...

%Partie lin aire
u0=D0*u;
u0=-c2*u0;
u2=D2*u;
ut =u0+u2;

%Partie non lin aire : int gration num rique
%src1(u) = u2.ux
yu2ux_1 = feval('integrand1_1',xquad')*wquad';
yu2ux_2 = feval('integrand2_1',xquad')*wquad';

%src2(u) = u4
yu4_1 = feval('integrand1_2',xquad',h(1))*wquad';
yu4_2 = feval('integrand2_2',xquad',h(1))*wquad';

%Assemblage de ut
ut(1:n-1,1) = ut(1:n-1,1)+ c1*yu2ux_1 + c2*yu4_1;
ut(2:n,1) = ut(2:n,1)+ c1*yu2ux_2+ c2*yu4_2;

%...
%Conditions aux limites
%...
ut(1) = ut(1)-(u(1)-u_anal(x0,t));
ut(n) = ut(n)-(u(n)-u_anal(xL,t));
```

```
end
```

integrand1_1.m

```
function out = integrand1_1(x)
global u1 n

for i = 1:length(x)
    [N1,N2] = trialfunctionslagr_1(x(i));
    [dN1dx,dN2dx] = der1trialfunctionslagr_1(x(i));

    u0x=N1*u1(1:n-1)+N2*u1(2:n);
    u0x=u0x.^2;
    u1x=dN1dx*u1(1:n-1)+dN2dx*u1(2:n);

    out(1:n-1,i) = N1*u0x.*u1x;
end
```

integrand1_2.m

```
function out = integrand1_2(x,h)
global u1 n

for i = 1:length(x)
    [N1,N2] = trialfunctionslagr_1(x(i));

    u0=(N1*u1(1:n-1)+N2*u1(2:n));
    u0=u0.^4;

    out(1:n-1,i) = (h/2)*N1*u0;
end
```

integrand2_1.m

```
function out = integrand2_1(x)
global u1 n

for i = 1:length(x)
    [N1 ,N2] = trialfunctionslagr_1(x(i));
    [dN1dx, dN2dx] = der1trialfunctionslagr_1(x(i));

    u0x=N1*u1(1:n-1)+N2*u1(2:n);
    u0x=u0x.^2;
    u1x=dN1dx*u1(1:n-1)+dN2dx*u1(2:n);

    out(1:n-1,i) = N2*u0x.*u1x;
end
```

integrand2_2.m

```
function out = integrand2_2(x,h)
global u1 n
```

```
for i = 1:length(x)
    [N1 ,N2] = trialfunctionslagr_1(x(i));

    u0=(N1*u1(1:n-1)+N2*u1(2:n));
    u0=u0.^4;

    out(1:n-1,i) = (h/2)*N2*u0;
end
```

6.3 Codes de la collocation orthogonale

Colloc.m

```
clear all
close all

%...
%Debut du chronometre
%...
tic

%...
%Variables globales
%...
global n x0 xL dx h D0 D2

%...
%Grille spatiale
%...
x0=-10;
xL=10;
n=501;
nel = n-1;
dx = (xL - x0)/(n-1);
x = (x0:dx:xL)';
h = (x(2:n) - x(1:n-1))';
hCI = ([h(1) h]+[h h(n-1)])/4;

%...
%Matrices de differentiation
%...
D0 = colorthD0(n);
D2 = colorthD2(n,h);
M = mcolorth(n,1);

%...
%Conditions initiales (u1,u1x,u2,u2x,u3,u3x,...)
%...
for i=1:n
    CI_0 (i) = u_anal(x(i),0);
```

```
    CI_X(i) = u_anal_der(x(i),0);
end
u(1:2:2*n-1) = CI_0;
u(2:2:2*n)= CI_X.*hCI;

%...
%Instants de visualisation
%...
dt=0.5;
t=(0:dt:5);
nt=length(t);

%...
%Integration temporelle
%...
options=odeset('Mass',M,'RelTol',1e-5,'AbsTol',1e-5,'stats','on');
%
%Solveurs differentiels-algebriques
[tout, yout] = ode15s(@BurgerFisherColloc,t,u,options);
%[tout, yout] = ode23t(@BurgerFisherColloc,t,u,options);

%...
%Fin du chronometre
%...
tcpu = toc

%...
%Affichage solution analytique VS solution numerique
%...

%Solution numerique
plot(x,yout(:,1:2:2*n-1),'.-k')
xlabel('x');
ylabel('u(x,t)');
title('Equation de Burger-Fisher : Collocation Orthogonale')
hold on

%Solution analytique
for k = 1:length(tout)
    for i = 1:n
        yexact(k,i) = u_anal(x(i),tout(k));
    end
    plot(x,yexact(k,:),'r')
    hold on
end

%...
%Calcul de l'erreur + Graphe de l'erreur
%...
```

```
figure;
e=ec(yexact,yout(:,1:2:2*n-1),x)
```

BurgerFisherColloc.m

```
function ut = BurgerFisherColloc(t,u)
%...
%Variables globales
%...
global n x0 xL D2 D0 u1 h
u1=u;
t
c1 = 2/((1+exp(-6*t))^(2/3));
c2 = (-2)/(1+exp(-6*t));

%...
%Construction de ut = -c2.D0.u + D2.u + c1.src1(u) + c2.src2(u)
%...

%Partie lineaire
ut = D2*u -c2*D0*u;

%Partie non lineaire : src evalue aux points de collocation
%src1(u) = u2.ux
yu2ux_1 = feval('integrand1',-1/sqrt(3),h(1));
yu2ux_2 = feval('integrand1',1/sqrt(3),h(1));

%src2(u) = u4
yu4_1 = feval('integrand2',-1/sqrt(3));
yu4_2 = feval('integrand2',1/sqrt(3));

%Assemblage de ut
ut(2:2:2*n-2,1) = ut(2:2:2*n-2,1)+ c1*yu2ux_1 + c2*yu4_1;
ut(3:2:2*n-1,1) = ut(3:2:2*n-1,1)+ c1*yu2ux_2+ c2*yu4_2;

%...
%Conditions aux limites
%...
ut(1,1) = u(1,1)-u_anal(x0,t);
ut(2*n,1) = u(2*n-1,1)-u_anal(xL,t);
end
```

integrand1.m

```
function out = integrand1(x,h)
%u2ux
global u1 n

[N1 ,N2 ,N3, N4] = trialfunctionsher_1(x);
[N1p, N2p, N3p,N4p] = der1trialfunctionsher_1(x);
```

```
u0 = N1*u1(1:2:2*n-3)+N2*u1(2:2:2*n-2)+N3*u1(3:2:2*n-1)+N4*u1(4:2:2*n);
u0=u0.^2;
ux = N1p*u1(1:2:2*n-3)+N2p*u1(2:2:2*n-2)+N3p*u1(3:2:2*n-1)+N4p*u1(4:2:2*n);

out(1:n-1,1) = (2/h)*u0.*ux;

end
```

integrand2.m

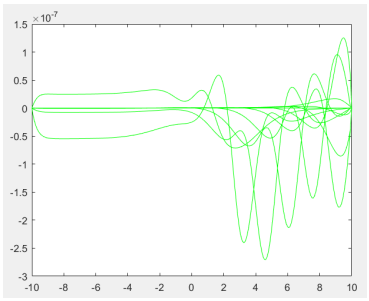
```
function out = integrand2(x)
%u4

global u1 n

[N1 ,N2 ,N3, N4] = trialfunctionsher_1(x);

u0 = N1*u1(1:2:2*n-3)+N2*u1(2:2:2*n-2)+N3*u1(3:2:2*n-1)+N4*u1(4:2:2*n);
u0=u0.^4;

out(1:n-1,1) = u0;
end
```

u_anal_der.m

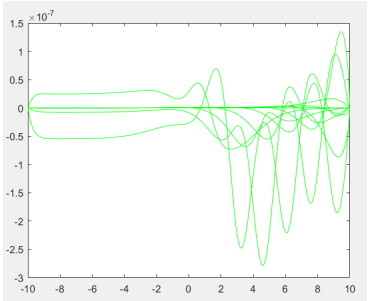
```
function output = u_anal_der(x,t)

output=((1+exp(-6*t))^(1/3))*((tanh((x-3*t)/2))^2-1)/4;

end
```

6.4 Graphiques des erreurs pour les différences finies (250 points)

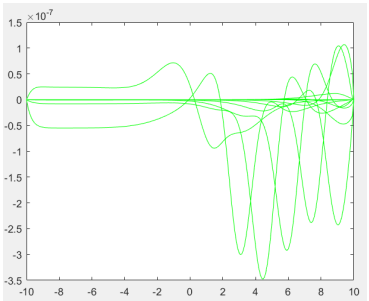
Les 24 figures suivantes montrent les graphes des erreurs pour le simulateur des différences finies en utilisant ode15s, option JPattern, 250 points, tolérances 10^{-7}. Les 12 premières utilisent le schéma $D2 = five\_point\_centered\_D2(x)$ tandis que la page d'après montre 12 autres figures qui utilisent le schéma $D2 = three\_point\_centered\_D2(x)$.



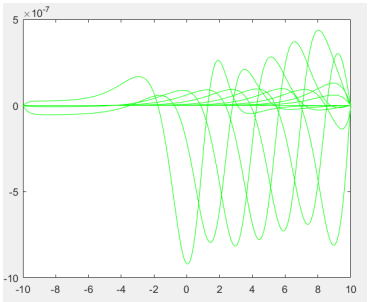
5 points biased upwind(1)



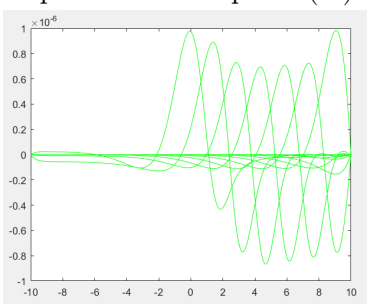
5 points biased upwind(-1)



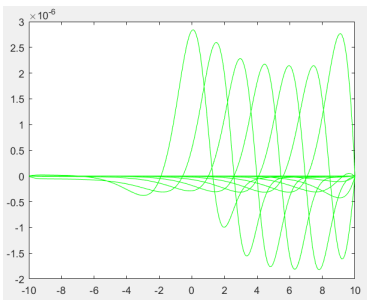
5 points centered



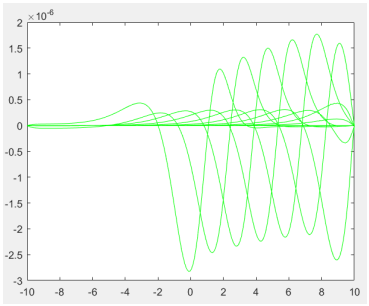
4 points biased upwind(1)



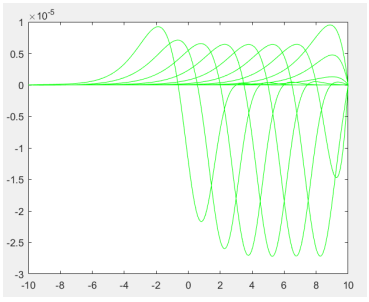
4 points biased upwind(-1)



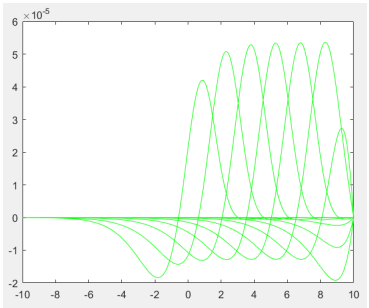
4 points upwind(1)



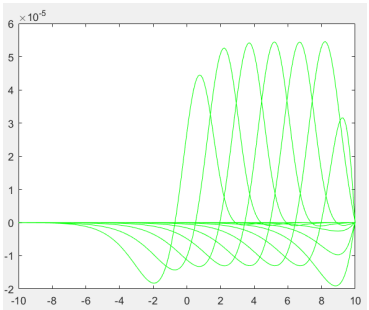
4 points upwind(-1)



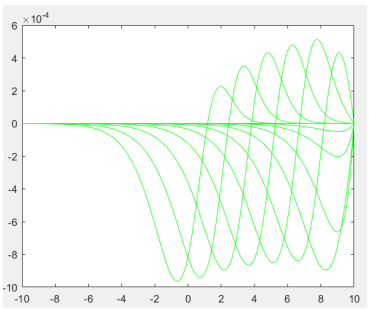
3 points centered



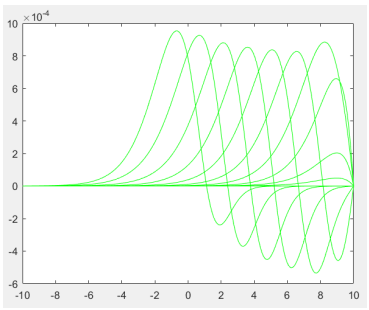
3 points upwind(1)



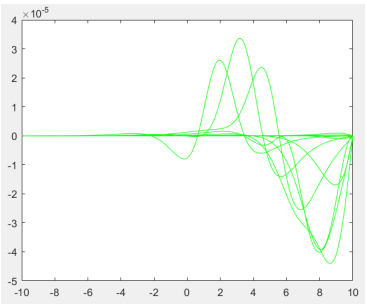
3 points upwind(-1)



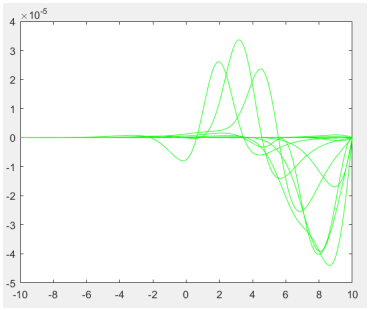
2 points upwind(1)



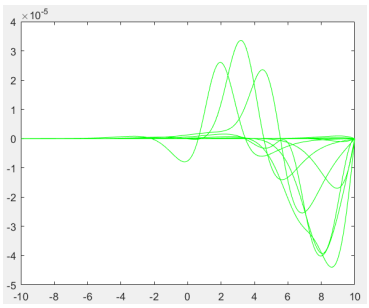
2 points upwind(-1)



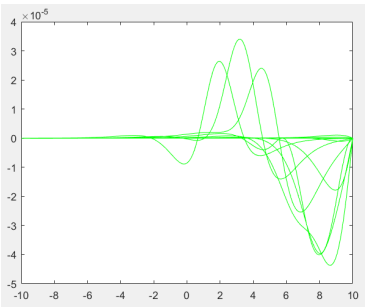
5 points biased upwind(1)



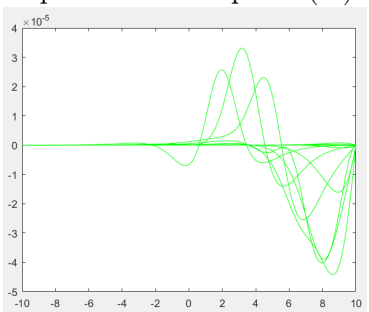
5 points biased upwind(-1)



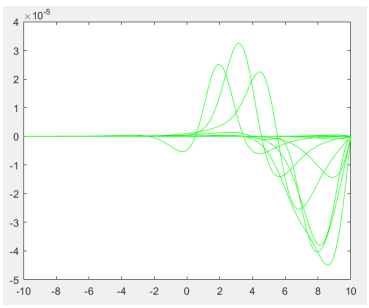
5 points centered



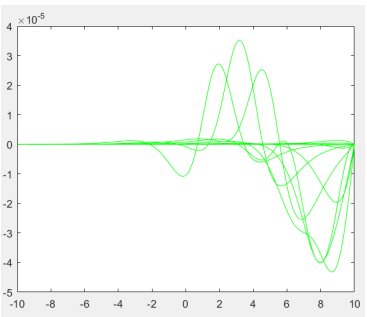
4 points biased upwind(1)



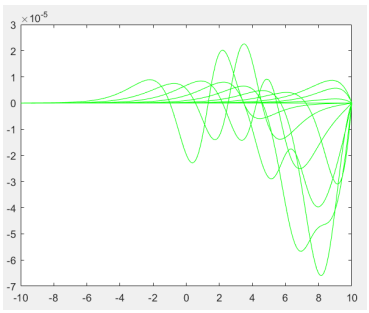
4 points biased upwind(-1)



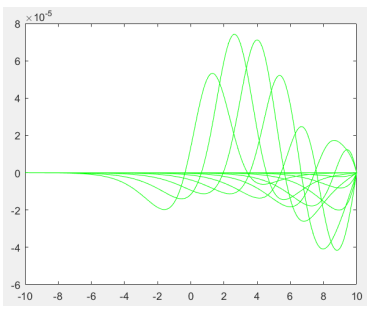
4 points upwind(1)



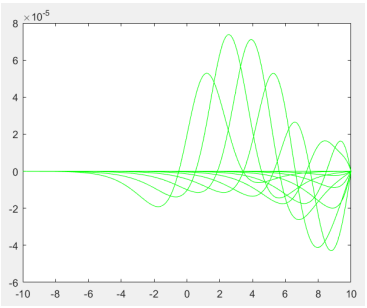
4 points upwind(-1)



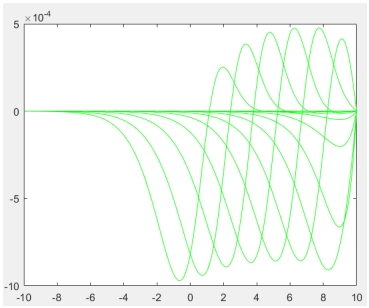
3 points centered



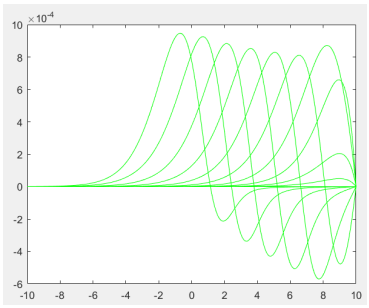
3 points upwind(1)



3 points upwind(-1)



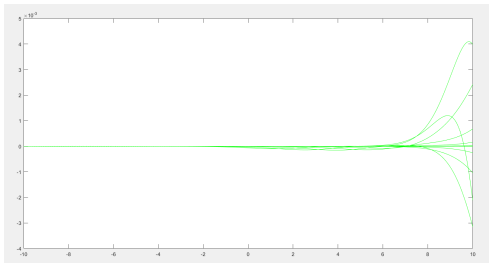
2 points upwind(1)

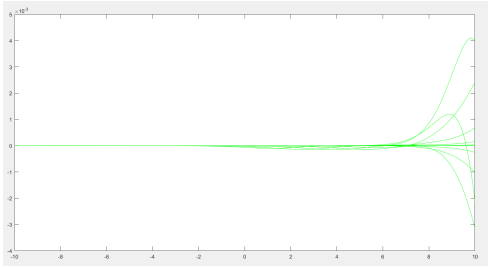


2 points upwind(-1)

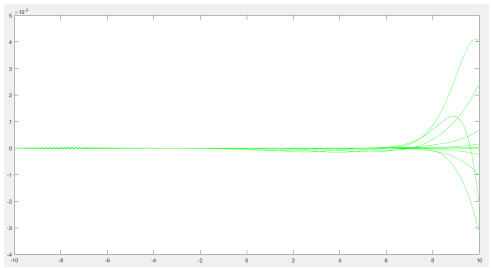
6.5 Graphiques des erreurs pour les éléments lagrangiens (250 points et tolérances 10^{-5})

Les figures suivantes mettent en vis-à-vis les graphiques d'erreurs des simuateurs lagrangiens. Parmi les paramètres à faire varier, nous pouvons choisir le nombre de points, les tolérances et l'intégrateur. Faire varier les deux premiers paramètres ne changent pas l'allure des courbes obtenues. Nous fixerons donc ces deux paramètres et observerons le résultat en fonction de l'intégrateur temporel choisi :

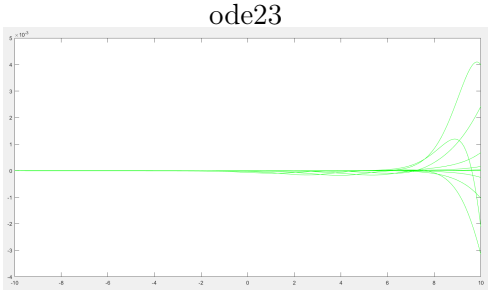




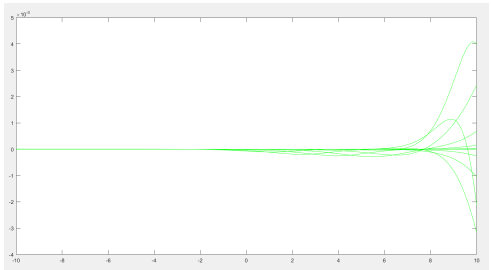
ode45



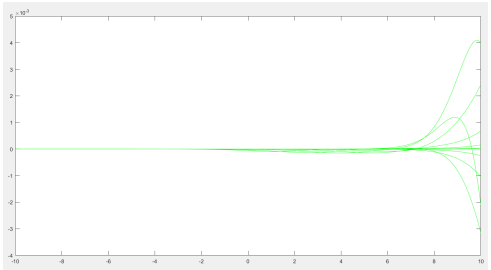
ode23



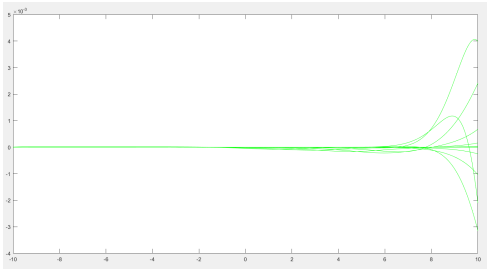
ode113



ode15s



ode23s



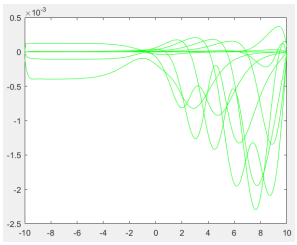
ode23tb

ode23t

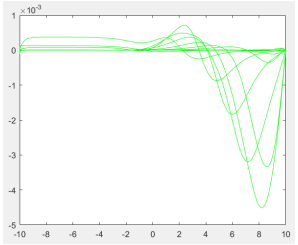
Il semblerait que même la variation des intégrateurs ne change pas l'allure et la répartition des erreurs sur le domaine spatial.

6.6 Graphiques des erreurs pour la collocation orthogonale (250 points)

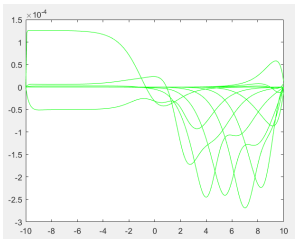
Les graphes ci-contre montrent l'influence du choix de l'intégrateur et des tolérances. Le choix du nombre de points a été fixé car il n'applique qu'un simple facteur d'échelle aux graphes.



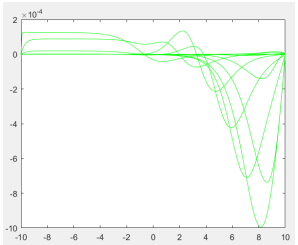
ode15s à 10^{-3}



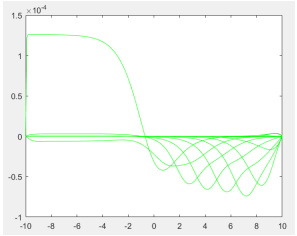
ode23t à 10^{-3}



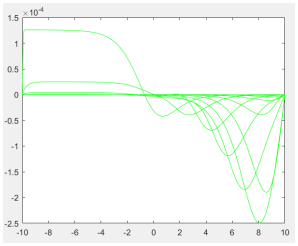
ode15s à 10^{-4}



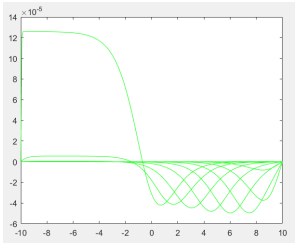
ode23t à 10^{-4}



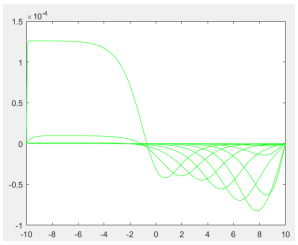
ode15s à 10^{-5}



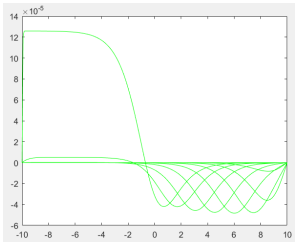
ode23t à 10^{-5}



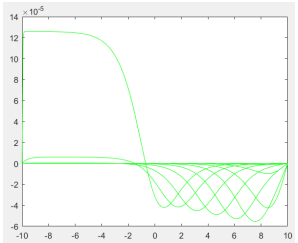
ode15s à 10^{-6}



ode23t à 10^{-6}



ode15s à 10^{-7}



ode23t à 10^{-7}