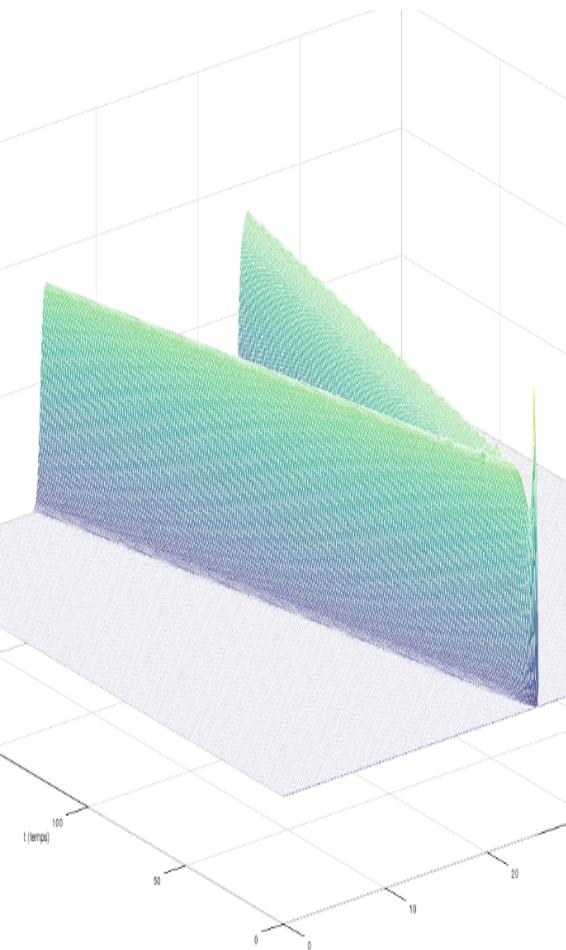


## Faculté Polytechnique

### Analyse numérique des équations aux dérivées partielles

Rapport de projet : Modèle d'impulsion dans une fibre nerveuse



Farid AFENZOUAR  
Mathis DELEHOUZEE  
Federico FISICARO  
Thomas GUILY



Dirigé par Monsieur  
Philippe SAUCEZ

Année académique 2018-2019

# Table des matières

<b>Énoncé du problème</b>	<b>2</b>
<b>1 Introduction</b>	<b>2</b>
<b>2 Construction du simulateur basé sur les différences finies</b>	<b>4</b>
2.1 Modélisation . . . . .	4
2.1.1 Script initial (Main.m) . . . . .	4
2.1.2 Fonction Impulse.m . . . . .	6
2.1.3 Visualisation de $v(z, t)$ . . . . .	7
2.2 Schémas de différences finies . . . . .	8
2.2.1 Nombre minimal de points, nombre de points de grille suffisant . . . . .	8
2.2.2 Influence du nombre de points dans l'approximation de la dérivée seconde .	11
2.2.3 Comparaison des outils d'intégration . . . . .	11
2.3 Méthode de calcul Jpattern . . . . .	12
2.4 Choix des outils pour le meilleur rapport qualité/temps de calcul . . . . .	12
<b>3 Implémentations de <math>S(z,t)</math></b>	<b>14</b>
3.1 Résultats pour Source.m . . . . .	15
3.2 Résultats pour Source1.m . . . . .	16
3.3 Résultats pour Source2.m . . . . .	16
<b>4 Generalités sur la méthode des éléments finis</b>	<b>20</b>
<b>5 Simulateur en éléments finis lagrangiens linéaires.</b>	<b>21</b>
5.1 Rappel théorique . . . . .	21
5.2 Modélisation . . . . .	21
5.2.1 Main . . . . .	21
5.2.2 Impulse . . . . .	23
5.2.3 Integrand1 . . . . .	24
5.2.4 Integrand2 . . . . .	24
5.2.5 Résultats . . . . .	24
<b>6 Simulateur en éléments finis hermitiens</b>	<b>27</b>
6.1 Rappel théorique . . . . .	27
6.2 Modélisation . . . . .	28
<b>Conclusion</b>	<b>31</b>
<b>7 Comparaison des méthodes</b>	<b>31</b>
7.1 Temps de calcul . . . . .	31
7.2 Allure des graphiques . . . . .	31
7.2.1 Hermite . . . . .	31
7.2.2 Lagrange . . . . .	32
7.2.3 Différences finies . . . . .	32

# 1 Introduction

De nos jour, la résolutions d'équations aux dérivées partielles représente un grand défi pour la recherche et l'innovation dans les calculs scientifiques. Ces équations permettent de décrire de nombreux phénomènes complexes afin de les modéliser efficacement. De très nombreux domaines dépendent de la mise en place et de la résolution de ces équations. De la mécanique des fluides à la chimie en passant par la biologie ou la physique, ces équations représentent un enjeux majeur à leur développement.

Notre projet consiste à modéliser la propagation d'une impulsion dans une fibre nerveuse au cours du temps à partir d'une EDP. Dans un premier temps, nous allons construire un simulateur en différence finies et discuterons de l'influence du choix des outils utilisés sur la qualité de la simulation et du temps de calcul. Par la suite, nous utiliserons l'option *Jpattern* des intégrateurs *ode* de Matlab afin d'étudier son influence sur notre système. Enfin, nous mettrons au point un simulateur en éléments finis lagrangiens et un autre en éléments finis hermitiens. Tout ceci nous fourniras une base solide pour la comparaison des différents modèles concernant leur qualité et leur efficacité.

Il est à noter que le code complet est disponible et téléchargeable sur github à l'adresse suivante : <https://github.com/ThomasGuily/BrainImpulse>

## Enoncé du problème

Ce projet modélise la propagation d'une impulsion dans une fibre nerveuse.

Cette propagation est décrite par une variante des équations de Fitzhugh-Nagumo :

$$\begin{aligned} v_t &= Dv_{zz} + v(v - 1)(-v) - w + S \\ w_t &= \beta_1 v - \beta_2 w \end{aligned}$$

Dans ces équations  $v(z, t)$  représente l'amplitude de l'impulsion,  $w(z, t)$  est une variable annexée et  $S(z, t)$  est un stimulus externe appliqué au milieu de la fibre. On a également :

$$0 \leq z \leq 50 \text{ ; } 0 \leq t \leq 200 \text{ ; } \Delta t = 0.2 \text{ ; } D = 0.01 \text{ ; } \mu = 0.08 \text{ ; } \beta_1 = 0.008 \text{ ; } \beta_2 = 2.54\beta_1$$

Le terme source prendra trois formes, quelque soit la forme de cette modélisation, le stimulus n'agit que les deux premières secondes de la simulation :

1) Modélisation ponctuelle simpliste en milieu de fibre :

$$\begin{aligned} S(z_M, t) &= 0.15 \text{ pour } 0 \leq t \leq 2 \\ S(z, t) &= 0 \text{ pour } 2 < t \end{aligned}$$

2) Il s'agit d'une implémentation plus réaliste, la source est toujours ponctuelle mais cette fois la quantité d'énergie fournie à la fibre reste constante :

$$\int_{(z_M - \frac{\Delta z}{2})}^{(z_M + \frac{\Delta z}{2})} S(z, t) dz = cte \text{ ; ce qui donne } S(z_M, t) = \frac{0.15}{\Delta z} \text{ pour } 0 \leq t \leq 2$$

Où  $z_m$  est la coordonnée du milieu de la fibre et  $\Delta z$  la distance séparant deux points de grille.

3) Il est illusoire de générer dans la pratique médicale un stimulus rigoureusement ponctuel. On élargit alors la plage des coordonnées sur laquelle le stimulus est programmé :

$$\int_{(z_M - k\Delta z - \frac{\Delta z}{2})}^{(z_M + k\Delta z + \frac{\Delta z}{2})} S(z, t) dz = cte \text{ ; ce qui donne } S(z_M - k : z_M + k, t) = \frac{0.15}{(2k+1)\Delta z} \text{ pour } 0 \leq t \leq 2$$

Les conditions initiales sont les suivantes :  $v(z, 0) = w(z, 0)$

Les conditions aux limites sont :  $v(z_{min}, t) = v(z_{max}, t) = 0$  ;  $w(z_{min}, t) = w(z_{max}, t) = 0$

## 2 Construction du simulateur basé sur les différences finies

Le principe des différences finies est d'approximer les dérivées temporelles et/ou spatiales d'une équation différentielle grâce au développement de Taylor autour d'un certain nombre de points. Ces méthodes peuvent être dite centrées (approximation en un point basée sur ce point et les points voisins) ou pas.

### 2.1 Modélisation

#### 2.1.1 Script initial (Main.m)

Voici le contenu du script initial (Main.m) :

```
close all  
clear all
```

On supprime tout ce qui a été créé.

#### Déclaration des variables

```
global z k B1 B2 D n mu D2 dz;
```

Ces variables globales pourront circuler de fichiers en fichiers sans devoir passer en paramètre.

#### Définition des paramètres

```
D = 0.01;  
mu = 0.08 ;  
k = 3;  
i=0;  
tmax= 200;  
pas=0.2;  
z0 = 0;  
zL = 50;  
n = 201;  
B1 = 0.008;  
B2 = 2.54*B1;
```

Ces paramètres sont ceux énoncés dans l'introduction,  $n$  est le nombre de point de grille.

#### Création de la grille spatio-temporelle

```
dz = (zL - z0)/(n - 1);  
z = z0:dz:zL;  
z = z';  
t=0:pas:tmax;  
t = t';
```

$z$  est un vecteur de taille  $n$  qui contient les différentes valeurs de  $z$  ( $0 \leq z \leq 50$ ).

$t$  est un vecteur de taille 1001 contenant les différentes valeurs temporelles ( $0 \leq t \leq 200$ ).

#### Approximation de la dérivée seconde

```
D2 = three_point_centered_D2(z);
```

La matrice de dérivée seconde selon  $z$  ( $v_{zz}$ ) est approximée selon la méthode des différences finies. Elle prend en entrée le vecteur  $z$ . Plusieurs méthodes d'approximation seront testées mais elles sont toutes centered (trois points, cinq points, sept points et neuf points).

## Vecteur des conditions initiales

```
v0 = zeros (length(z),1);  
w0 = zeros (length(z),1);  
u0 = [v0;w0];
```

Les deux vecteurs de conditions initiales sont rassemblés dans un seul et même vecteur  $u0$  qui rentrera ensuite dans la fonction Ode.

## Initialisation de Ode

```
options=odeset ('RelTol',1e-5,'AbsTol',1e-5,'stats','on');
```

Ici l'option Jpattern n'est pas utilisée mais elle le sera par la suite. On peut voir que l'erreur relative et l'erreur absolue sont fixées à  $1e^{-5}$ .

## Lancement du chronomètre

```
tic
```

## Appel de Ode

```
[tout , yout] = ode45(@Impulse,t,u0,options);
```

Il s'agit ici de Ode45, mais plusieurs intégrateurs seront testés. Il est important de signaler que cette fonction utilise une autre fonction que nous avons modélisé (ici Impulse.m). Ode prend en entrée  $t$ ,  $u0$  (vecteur de conditions initiales) et la fonction en question ainsi que les paramètres choisis.  $yout$  contiendra toutes les valeurs de  $u$  calculées à partir de  $u0$  et ce pour chaque itérée de  $t$ .

## On ne garde que $v$

```
yout = yout (:,1 :length(z));
```

On laisse tomber  $w$  pour la représentation graphique puisque seul  $v$  nous intéresse.

## Arrêt et lecture du chronomètre

```
tcpu=toc;  
tcpu
```

On récupère le temps de calcul afin d'observer les performances du modèle.

## Visualisation graphique

```
Visualizer(z,t,yout);
```

La fonction Vizualizer.m permet d'afficher les valeurs de  $v$  pour chaque points de la grille spatio-temporelle. Elle sera décrite un peu plus loin.

### 2.1.2 Fonction Impulse.m

La fonction Impulse.m permet de calculer une nouvelle itérée de  $v_t$  et donc de  $w_t$  par la même occasion via l'utilisation de la fonction Ode. Voici son contenu :

```
function [ ut ] = Impulse(t, u)
```

Ici on peut voir que la fonction prend en input la valeur de  $u$  pour un  $t$  fixé. Lors de l'appel de Ode, à la première itération, on a  $u = u_0$  et  $t = 0$ , la fonction va calculer  $ut$ , l'intégrateur va reprendre cette valeur en sortie et calculer la valeur suivante de  $u$  pour passer à nouveau dans cette fonction. De proche en proche, la matrice  $y_{out}$  va stocker toutes les valeurs de  $u$  calculées.

```
global z D D2 mu B1 B2;
```

Variables globales utilisées dans cette fonction.

### Calcul du terme source

```
S = Source(z, t);
```

Avant de calculer cette itérée il est important de calculer  $S$  selon l'un des trois modèles énoncé. Il faut donc faire appel à l'une des trois fonctions (Source.m, Source1.m ou Source2.m) qui modélisent respectivement et dans l'ordre de l'énoncé les trois termes sources qui seront étudiés.

### Séparation du vecteur initial en $v$ et $w$

```
v = u(1:length(z))';  
w = u(length(z)+1:2*length(z));
```

On sépare  $u$  en  $v$  et  $w$ .

### Conditions aux limites

```
v(1) = 0;  
v(length(z)) = 0 ;  
w(1) = 0;  
w(length(z)) = 0 ;
```

Voir énoncé.

### Calcul d'une itérée

```
ut = [D*D2*v' + (v.* (v-1).* (mu-v))' - w' + S; (B1*v - B2*w)'];
```

On range directement l'itérée calculée de  $v_t$  et celle de  $w_t$  dans  $u_t$ .

### 2.1.3 Visualisation de $v(z, t)$

```
function Visualizer(z,t,yout)
```

Cette fonction prend en input  $z, t$  et  $y_{out}$  qui contient chacune des valeurs de  $v(z, t)$  pour la grille utilisée.

```
clf;
```

On ferme les figures déjà ouvertes.

### Création de la figure

```
mesh(z , t , yout);
title ('Amplitude du signal source appliquee entre t=0 et t=2');
xlabel 'z (position)';
ylabel 't (temps)';
zlabel 'v (amplitude du signal)';
```

La fonction mesh permet de créer un grillage et place celui-ci dans un graphique en trois dimensions. D'autres textures de grillage existent (mechc,...) et on peut changer la palette de couleur.

Voici un exemple d'un résultat obtenu lors de l'affichage :

**Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$**

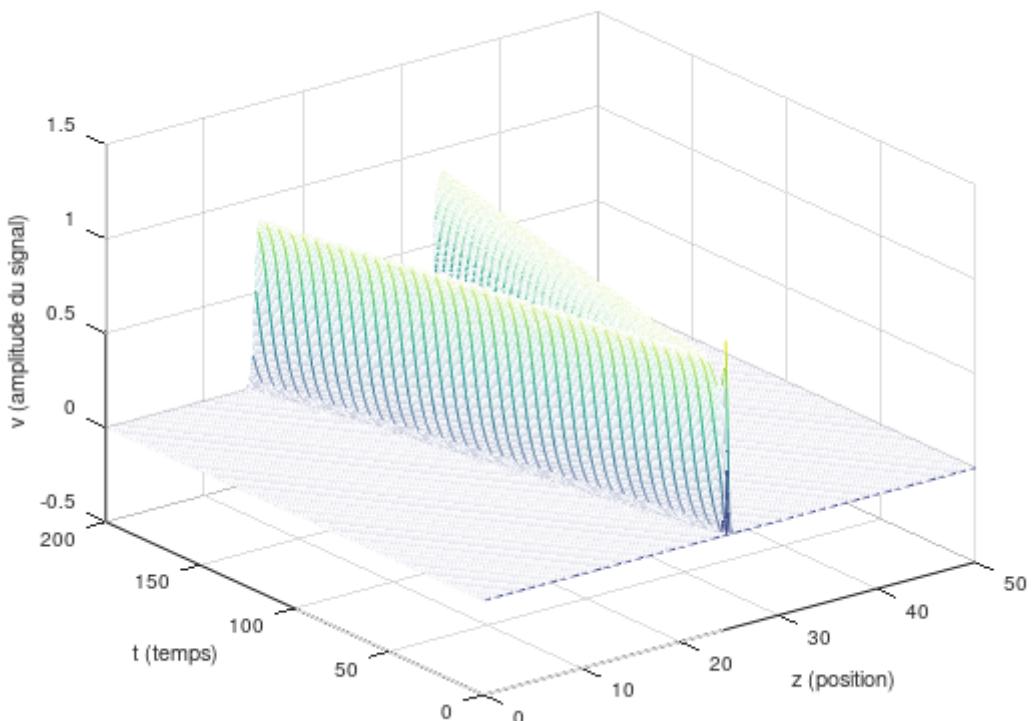


FIGURE 1 – Exemple pour Source1.m et  $n = 201$

Il est important de préciser, pour ce graphique et pour tout ceux qui suivent, que à cause de la perspective de visualisation on dirait que l'impulsion soit placée au delà de  $z=30$ . Ce pendant lorsque on se déplace dans l'espace 3D d'affichage, à fin de se mettre face au plan ( $v, z$ ), on voit que le pic se trouve bien en  $z=25$ .

## 2.2 Schémas de différences finies

### 2.2.1 Nombre minimal de points, nombre de points de grille suffisant

On laisse de coté la vitesse d'exécution afin de trouver le nombre de points de grille suffisant à une simulation qualitative.

Utilisons d'abord la source initiale. Celle-ci n'apporte pas une énergie constante quel que soit le nombre de points de grille, autrement dit, les graphiques obtenus n'auront pas la même forme quand  $n$  varie. Ce terme source n'est donc pas optimal afin de déterminer le nombre suffisant de points de grille, par contre, on peut observer certaines variations des profils graphiques en fonctions de  $n$  :

La première source possède trois profils caractéristiques :

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

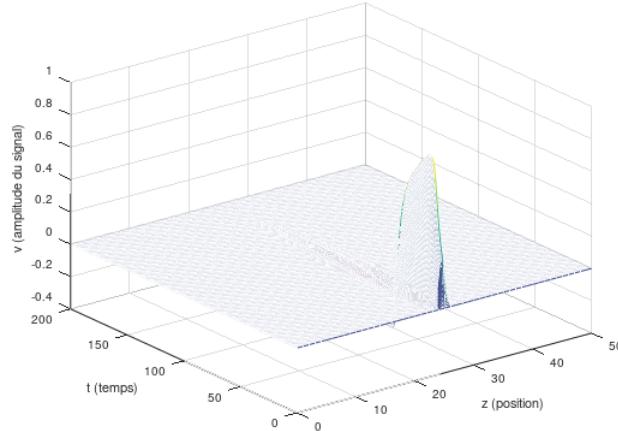


FIGURE 2 – Source.m,  $n = 51$

Avant  $n \approx 75$  le signal ne se propage pas jusqu'à  $t = 200$ , le nombre de points est insuffisant, on observe un pic allongé (cela est valable pour de petites valeurs de  $n$ ).

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

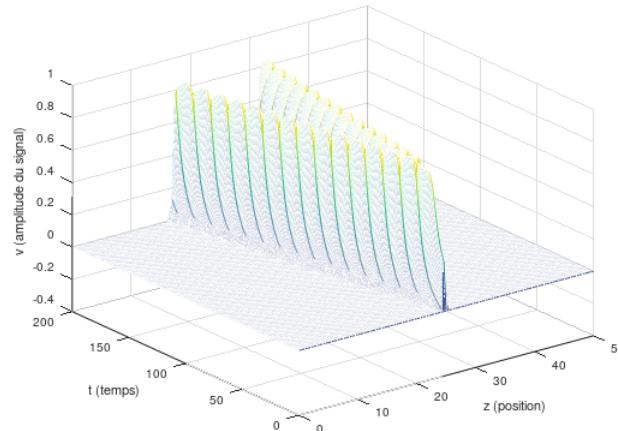
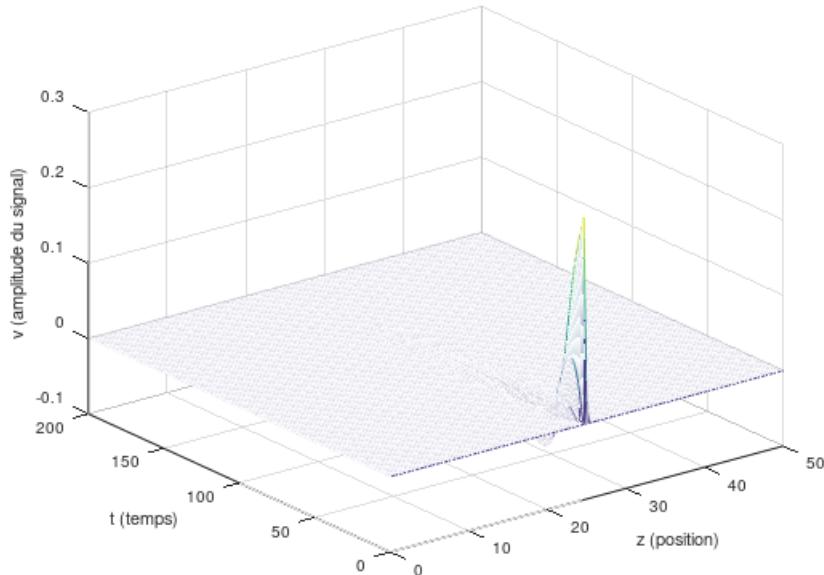


FIGURE 3 – Source.m,  $n = 101$

Entre  $n \approx 75$  et  $n \approx 115$ , le signal se propage de bout en bout du domaine temporel et forme deux raies qui s'éloignent l'une de l'autre. Le nombre de points n'est toujours pas suffisant pour obtenir un résultat qualitatif.

**Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$**



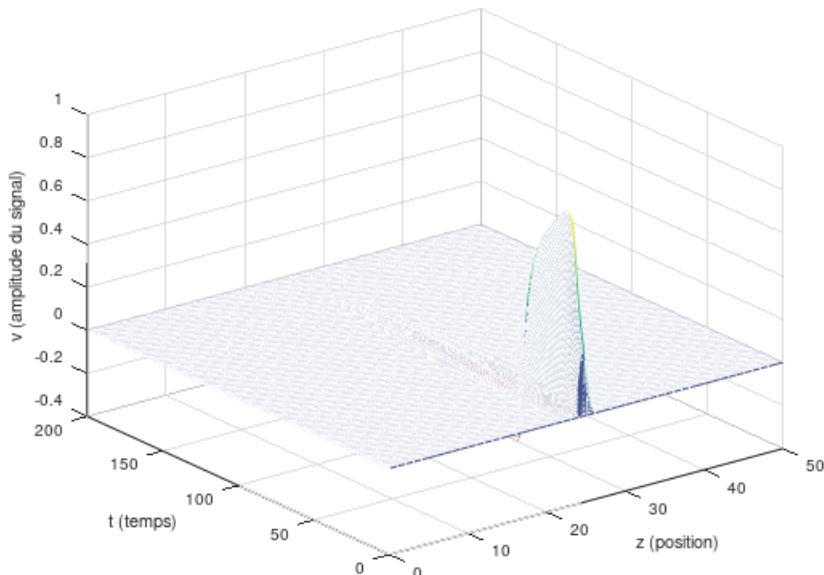
**FIGURE 4 – Source.m,  $n = 151$**

Au dessus de  $n \approx 115$ , le signal ne se propage plus et forme un pic moins allongé. En augmentant  $n$ , le pic d'intensité diminue, car l'énergie apportée à la fibre est moindre.

On peut en conclure que entre  $n \approx 75$  et  $n \approx 115$  l'énergie apportée à la fibre est suffisante pour que le signal se propage mais le nombre de point est insuffisant pour permettre une simulation qualitative. Lorsque l'énergie apportée à la fibre descend sous un certain seuil, le signal ne se propage plus. Le nombre de points sera suffisant à partir d'un certain moment mais ne représentera qu'un simple pic, ce qui n'est pas très intéressant.

Utilisons maintenant la deuxième source, pour laquelle l'énergie apportée à la fibre est constante :

**Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$**



**FIGURE 5 – Source1.m,  $n = 51$**

Avant  $n \approx 75$  le signal ne se propage pas jusqu'à  $t = 200$ , le nombre de points est insuffisant, la figure est d'ailleurs identique à celle obtenue avec la source initiale.

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

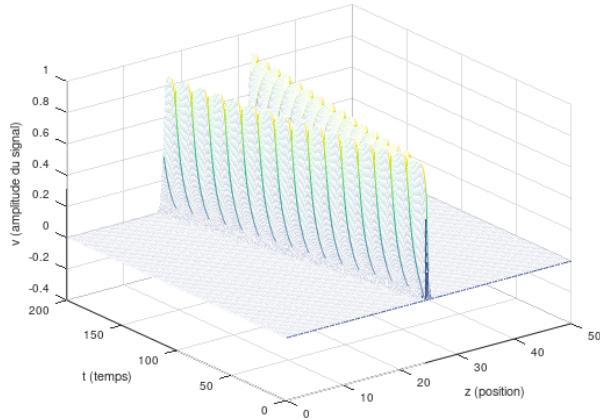


FIGURE 6 – Source1.m,  $n = 101$

Au dessus de  $n \approx 75$  le signal se propage correctement, mais le nombre de points est insuffisant.

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

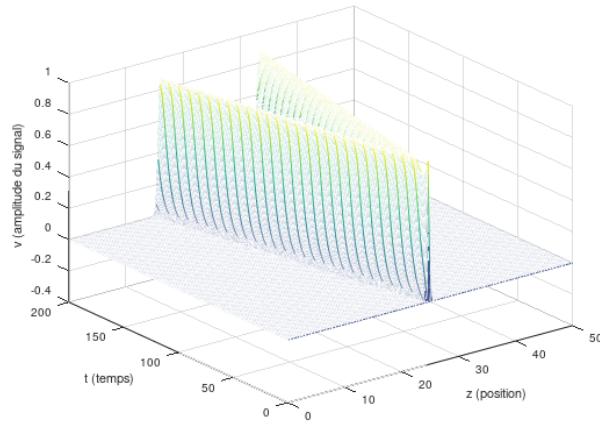


FIGURE 7 – Source1.m,  $n = 151$

Apparition d'un pic en début de simulation,  $n$  est insuffisant car ce pic n'est presque pas visible.

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

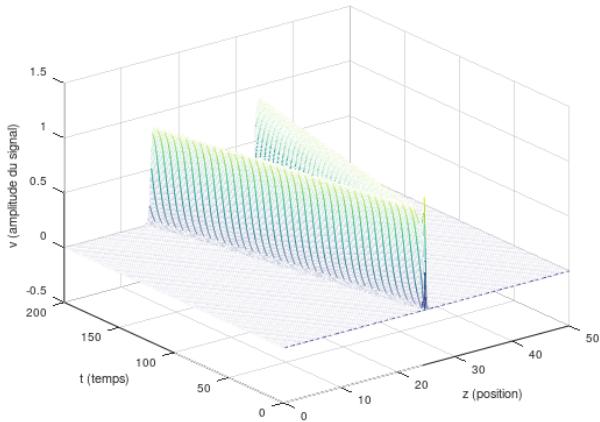


FIGURE 8 – Source1.m,  $n = 201$

A partir de  $n \approx 201$  la simulation devient qualitative (il n'y a presque plus de différences quand  $n$  augmente, on observe bien le pic), le nombre de point est donc suffisant. Pour la suite, nous prendrons  $n = 401$  pour avoir des résultats précis, de plus, le temps de calcul reste très acceptable (une seconde seulement avec Ode45 pour 401 points).

## 2.2.2 Influence du nombre de points dans l'approximation de la dérivée seconde

On fixe  $n = 401$ , on utilise la deuxième source et l'intégrateur est ode45 :

On teste quatre méthodes, les schémas sont centrés, on désire approximer la matrice de dérivée seconde  $v_{zz}$  (notons que, plus un schéma utilise de points, plus celui-ci minimise l'erreur et fournit donc un résultat de meilleure qualité.). Les méthodes utilisées sont celles à trois points, cinq points, sept points et neuf points. Les résultats sont tous corrects et semblables pour chaque méthode. De plus, les temps de calcul sont sensiblement les mêmes (dans l'ordre : 1.1409s, 1.2080s, 1.2387s et 1.3169s). On utilisera  $D2 = \text{ninepointcenteredD2}(z)$  pour la suite des opérations, l'erreur sur le terme de la dérivée seconde n'est alors plus proportionnelle qu'au cube de  $dz$  (cf syllabus) :

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

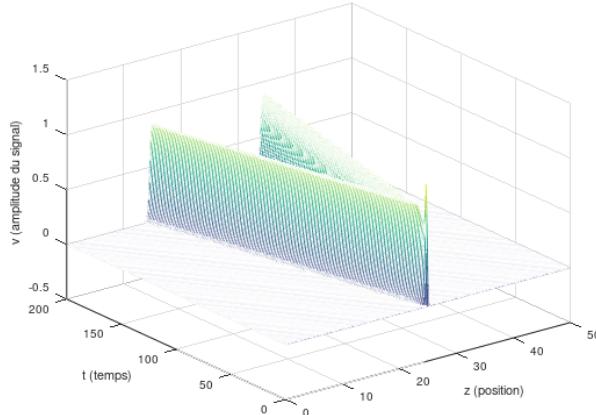


FIGURE 9 – Source1.m avec le schéma à neuf points centrés,  $n = 401$

## 2.2.3 Comparaison des outils d'intégration

On fixe  $n = 401$ , on utilise la deuxième source et le schéma centré à neuf points :

Voici un tableau reprenant les caractéristiques de chacun des intégrateurs temporels que nous allons tester :

Intégrateur	Type d'intégrateur	JPattern	Ordre de précision
ODE 45	Non Stiff	Non	Moyen
ODE 23	Non Stiff	Non	Faible
ODE 23s	Stiff	Oui	Faible
ODE 23t	Stiff	Oui	Faible
ODE 23tb	Stiff	Oui	Faible
ODE 113	Non Stiff	Non	Faible à élevé
ODE 15s	Stiff	Oui	Faible à moyen

Voici les résultats obtenus :

Intégrateur	Temps de calcul
ODE 45	1.3875s
ODE 23	1.0183s
ODE 23s	340.9804s
ODE 23t	10.4758s
ODE 23tb	14.3193s
ODE 113	1.5718s
ODE 15s	7.4196s

Les profils sont similaires et corrects pour chacun des intégrateurs temporels de la liste.

En toute logique, les intégrateurs non stiff sont les plus rapides pour notre type de problème puisqu'il est lui aussi non stiff et que l'option Jpattern est inutilisable.

On gardera Ode 45 pour la suite, car il reste rapide et il s'agit de l'intégrateur non stiff le plus précis.

Pour le choix du meilleur intégrateur, Ode23 remporte la bataille en étant le plus rapide et en restant très précis pour notre problème.

### 2.3 Méthode de calcul Jpattern

La méthode de calcul Jpattern n'a pas lieu d'être dans le cadre des ODE non stiff, elle n'est donc pas exploitable pour notre projet (cela a été vérifié par l'expérience et ce pour les trois sources). Pire, la Jacobienne est tout de même stockée dans la fonction Ode, ce qui ralenti le temps de calcul surtout pour un grand nombre de points de grille.

Pour autant, nous avons pris le temps de calculer la Matrice Jacobienne de ce problème :  
Puisque il s'agit d'un système à deux variables ( $v$  et  $w$ ), on a :

$$J_B = \begin{bmatrix} \frac{\partial F_1}{\partial v} & \frac{\partial F_1}{\partial w} \\ \frac{\partial F_2}{\partial v} & \frac{\partial F_2}{\partial w} \end{bmatrix}$$

Ou  $F_1$  correspond à l'équation en  $v_t$  et  $F_2$  correspond à l'équation en  $w_t$ .

On développe :

$$\frac{\partial F_1}{\partial v} = \begin{bmatrix} D_{11} - 3v_1^2 + 2(1 + \mu)v_1 - \mu & D_{12} & \dots & D_{1n} \\ D_{21} & D_{22} - 3v_2^2 + 2(1 + \mu)v_2 - \mu & \dots & D_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ D_{n1} & D_{n2} & \dots & D_{nn} - 3v_n^2 + 2(1 + \mu)v_n - \mu \end{bmatrix}$$

$$\frac{\partial F_1}{\partial w} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}; \frac{\partial F_2}{\partial v} = \begin{bmatrix} \beta_1 & 0 & \dots & 0 \\ 0 & \beta_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \beta_1 \end{bmatrix}; \frac{\partial F_2}{\partial w} = \begin{bmatrix} -\beta_2 & 0 & \dots & 0 \\ 0 & -\beta_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\beta_2 \end{bmatrix}$$

Ou les termes  $D_{ii}$  sont les termes de la matrice D2. Cela donnerait, au niveau du code, pour les options de ODE :

```
options=odeset('RelTol',1e-5,'AbsTol',1e-5,'stats','on','jpattern',
    sparse (spones([eye(n) + spones(D2), eye(n); eye(n), eye(n)])));
```

Mais cette option est inutile puisque le problème est non stiff. Cela se vérifie d'ailleurs en comparant les temps de calcul.

### 2.4 Choix des outils pour le meilleur rapport qualité/temps de calcul

On choisi donc :

- Nombre de points de grille :  $n = 401$ .
- Schéma : centré à neuf points.
- Intégrateur temporel : Ode23.

- Options : par défaut.

Voici le résultat :

**Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$**

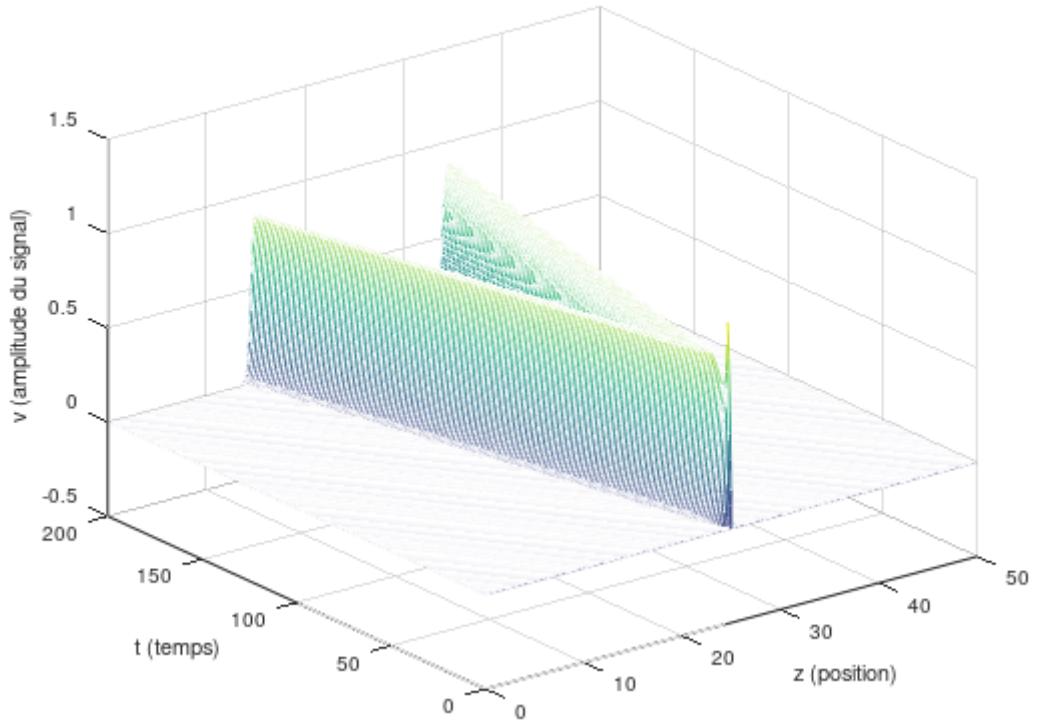


FIGURE 10 – Source1.m,  $n = 401$ , ode23, neuf points centrés

On observe qu'il est toujours très précis même en utilisant Ode23.

### 3 Implémentations de $S(z,t)$

#### A) function $S = \text{Source}(z,t)$

Il s'agit ici du terme source initial de l'énoncé.

##### Initialisation du vecteur source

```
S = zeros(length(z),1);
```

Initialisation de  $S$  comme étant un vecteur rempli de zéros de taille  $n$ .

##### Terme source initial, source ponctuelle

```
if t <= 2
    S(floor(length(z)/2)+1)=0.15;
end
```

Si  $t$  est inférieur à deux, alors  $S(z = 25) = 0.15$

#### B) function $S = \text{Source1}(z,t)$

Deuxième terme source, l'énergie injectée dans la fibre est constante, la source reste ponctuelle.

```
global dz;
```

On a besoin de  $dz$  qui est justement une variable globale.

##### Initialisation du vecteur source

```
S = zeros(length(z),1);
```

Initialisation de  $S$  comme étant un vecteur rempli de zéros de taille  $n$ .

##### Deuxième terme source, source ponctuelle améliorée

```
if t <= 2
    S(floor(length(z)/2)+1)= (0.15/dz) ;
end
```

Si  $t$  est inférieur à deux, alors  $S(z = 25) = \frac{0.15}{dz}$

#### C) function $S = \text{Source2}(z,t)$

Elargissement du spectre d'application de la source, l'énergie apportée à la fibre est toujours constante.

```
global k dz;
```

$k$  est un nombre entier, plus il est grand, plus le nombre d'entrées de  $S$  non nulles sera grand.

## Initialisation du vecteur source

```
S = zeros(length(z),1);
W = (2*k + 1)*dz ;
```

Initialisation de  $S$  comme étant un vecteur rempli de zéros de taille  $n$ .

## Troisième terme source, élargissement du stimulus

```
if t <= 2
    S(floor(length(z)/2)+1 - k : floor(length(z)/2)+1 + k) = (0.15/W) ;
end
```

Ici on perd la caractère ponctuel, on insérera la valeur  $\frac{0.15}{(2k+1)dz}$  dans les cases  $S(z_M - k)$  à  $S(z_M + k)$ .

### 3.1 Résultats pour Source.m

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

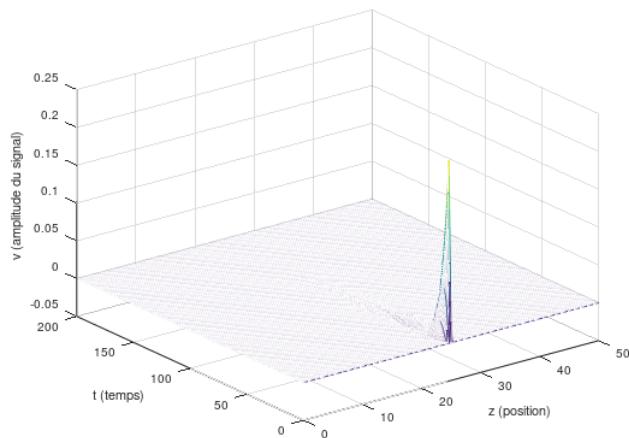


FIGURE 11 – Source.m,  $n = 201$

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

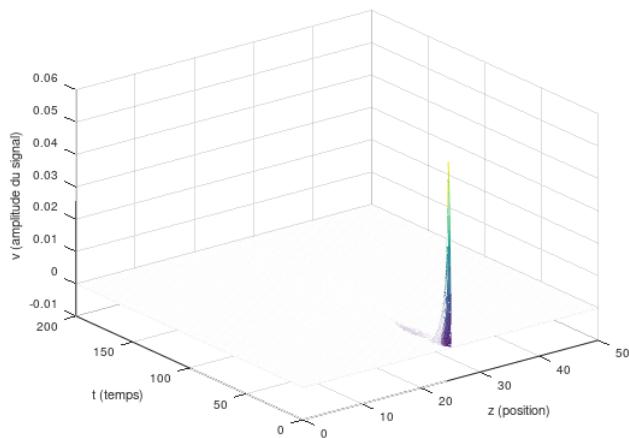


FIGURE 12 – Source.m,  $n = 1001$

On peut observer un changement d'intensité du signal en fonction du nombre de points de grille, le problème est non stiff. De plus, lorsque le nombre de points de grille augmente, l'énergie fournie à la fibre diminue, cela explique la diminution du pic sur la figure 12. Des tests ont été effectués jusqu'à 6501 points de grille, l'amplitude du signal n'est alors plus que de 0.008.

### 3.2 Résultats pour Source1.m

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

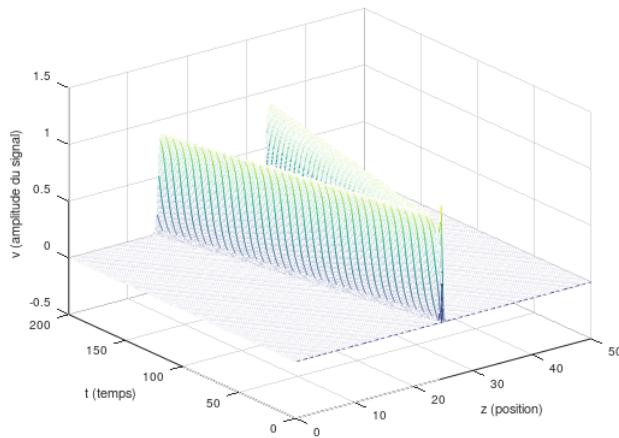


FIGURE 13 – Source1.m,  $n = 201$

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

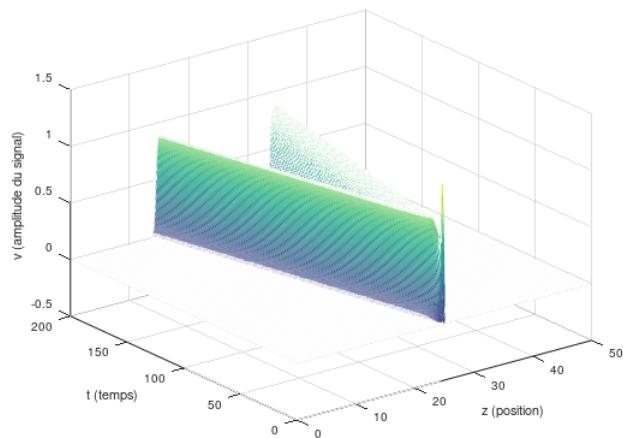


FIGURE 14 – Source1.m,  $n = 1001$

Cette fois les deux simulations sont presque identiques (si la grille n'est pas trop fine). Cela est logique, l'énergie injectée étant constante et ce quelque soit le nombre de points de grille.

### 3.3 Résultats pour Source2.m

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

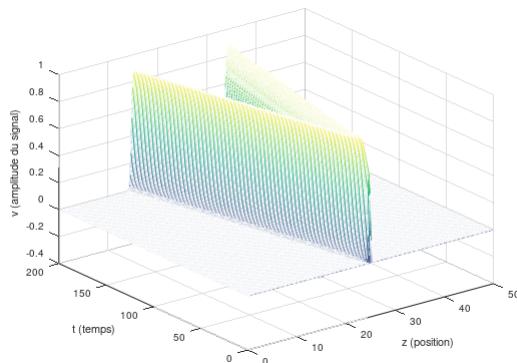


FIGURE 15 – Source2.m,  $n = 301$ ,  $k = 1$

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

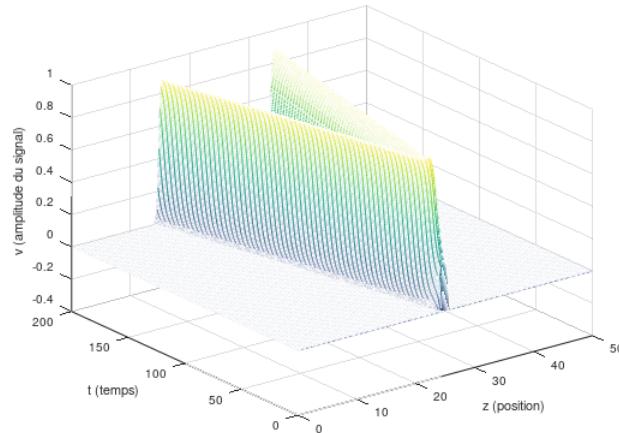


FIGURE 16 – Source2.m,  $n = 301$ ,  $k = 3$

Lorsque la finesse du maillage est faible, changer la valeur de  $k$  (autrement dit l'étalement du stimulus) change le résultat obtenu.

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

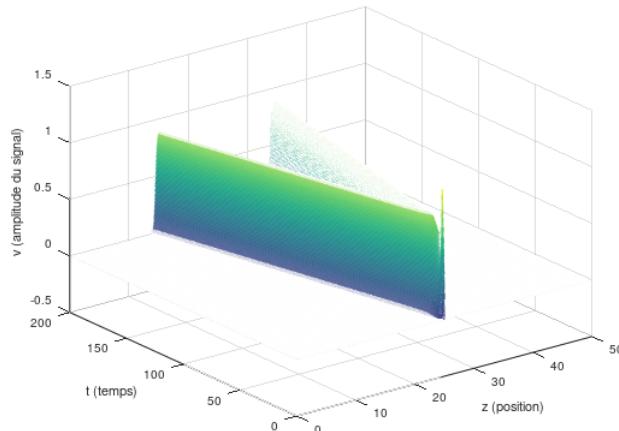


FIGURE 17 – Source2.m,  $n = 2001$ ,  $k = 1$

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

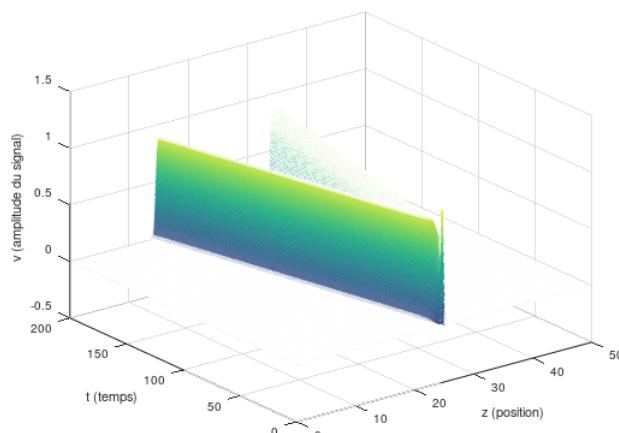


FIGURE 18 – Source2.m,  $n = 2001$ ,  $k = 3$

Lorsque la grille est très fine (2001 points ici), la valeur de  $k$  devient strictement indépendante du résultat. On obtient alors des figures semblables.

Fixons  $k = 5$  et faisons varier le nombre de points de grille :

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

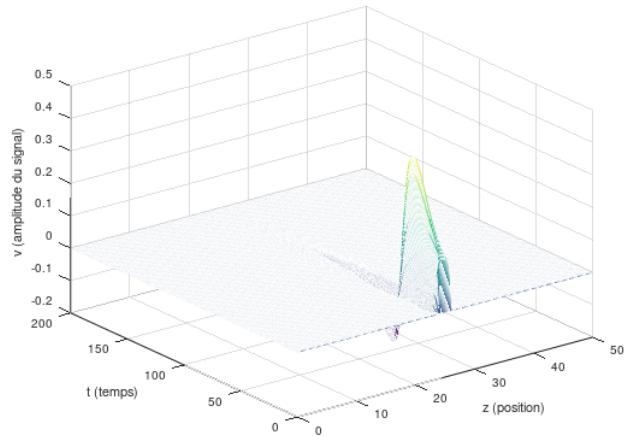


FIGURE 19 – Source2.m,  $n = 285$ ,  $k = 5$

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

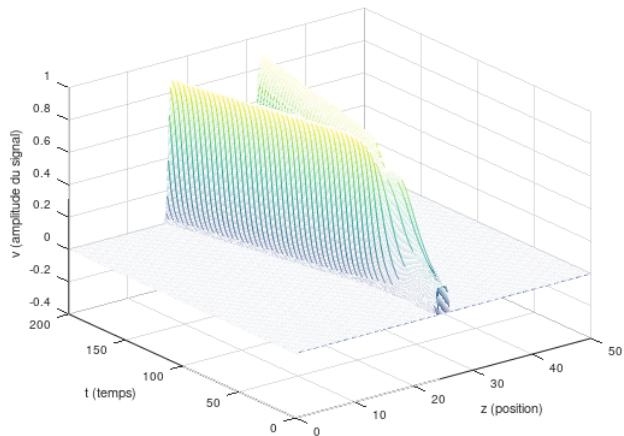


FIGURE 20 – Source2.m,  $n = 286$ ,  $k = 5$

On remarque ici une différence notable entre les deux profils alors qu'un seul et unique point de grille a été ajouté.

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

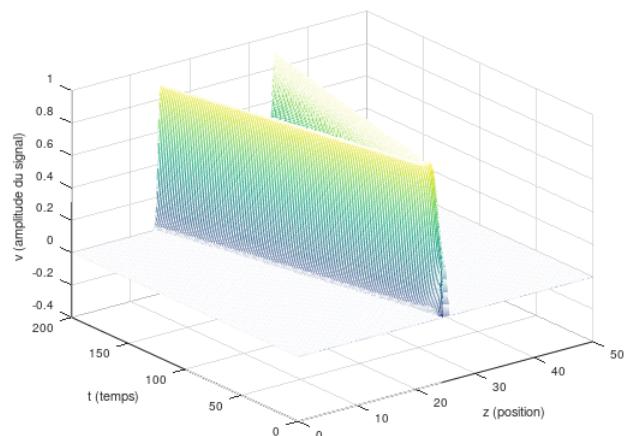


FIGURE 21 – Source2.m,  $n = 501$ ,  $k = 5$

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

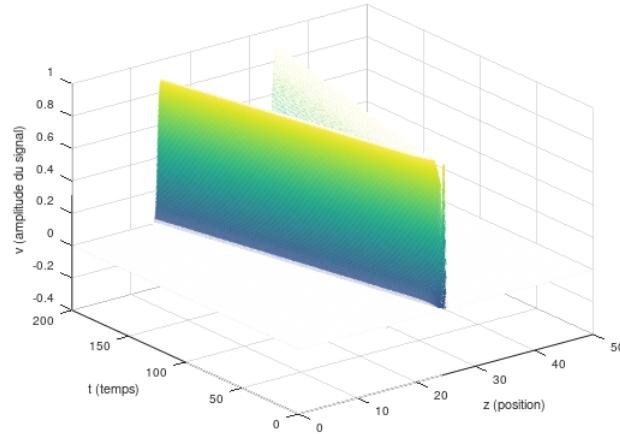


FIGURE 22 – Source2.m,  $n = 2001$ ,  $k = 5$

On voit apparaître un pic petit à petit.

Il y a donc une frontière délimitant deux zones. On remarque que cette frontière réapparaît quel que soit la valeur de  $k$ . En fait, elle se situe vers un rapport  $\frac{k}{n} \approx 0.018$  (pour  $k$  plus grand on tend vers 0.019, pour un  $k$  plus petit on tend vers 0.017).

On distingue alors deux zones caractéristiques, la zone où le stimulus est trop étalé par rapport au nombre de points de grille, on y trouve une forme de pic et le signal ne se propage pas jusqu'au bout de la simulation ( $\frac{k}{n} > 0.018$ ). Cela explique les résultats en forme de pic pour de petites valeurs de  $n$  avec les deux premières sources (malgré le fait que l'application du stimulus soit ponctuelle, le nombre de point de grilles est trop petit, le signal ne se propage pas).

On observe une seconde zone où le signal évolue en deux raies et se propage au bout de la simulation ( $\frac{k}{n} < 0.018$ ).

Notons que lorsque ce rapport tend vers zéro, on se rapproche de la simulation utilisant Source1.m (deuxième terme source) et on obtient un pic en début de simulation.

## 4 Generalités sur la méthode des éléments finis

La méthode des éléments finis consiste à substituer la fonction inconnue  $u(x, t)$  par :

$$\hat{u} = \sum u_k(t)N_k(x)$$

où les  $N_k$  sont appelés fonctions de base, dont le choix est arbitraire, et les  $u_k$  les valeurs inconnues de la fonction  $u$  au point de coordonnée  $x_k$ . Remplacer  $u$  par  $\hat{u}$  entraîne bien évidemment des erreurs, qu'il va falloir minimiser. À ce but on applique la méthode des résidus pondérés, qui consiste à égaler à 0 une série d'intégrales de la forme :

$$\int_{\Omega} W_l R_{\Omega} d\Omega \quad l = 1, 2, \dots, S$$

Ici les  $W_l$  sont appelées fonctions de pondération et, comme pour les  $N_k$ , leur choix est arbitraire.  $R_{\Omega}$  est par contre le résidu obtenu en remplaçant  $u$  par  $\hat{u}$  dans l'équation de départ (cette dernière valant normalement 0). Autre étape cruciale de cette méthode est le découpage du domaine  $\Omega$  en éléments de longueur finie  $\Omega_1, \Omega_2, \dots, \Omega_E$ , tel que chaque  $\Omega_i$  soit compris entre  $x_{i-1}$  et  $x_i$ . Ce ci permettra d'effectuer le calcul sur un seul élément au lieu que sur l'entièreté du domaine  $\Omega$ . Dans notre cas nous nous trouvons devant un système composé de 2 EDP sous la forme :

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

Où, dans  $F()u$ , les 3 premiers termes de droite constituent la partie linéaire de tandis que le dernier terme "src(u)" désigne le terme source. Ce dernier terme nécessitera un traitement numérique étant donné sa forme non-linéaire.

En appliquant la méthode des résidus pondérés sur ce type d'équation on obtient donc :

$$\int_{\Omega(x)} W_l \hat{u}_t dx = \int_{\Omega(x)} W_l \hat{u} dx + \int_{\Omega(x)} W_l \hat{u}_x dx + \int_{\Omega(x)} W_l \hat{u}_{xx} dx + \int_{\Omega(x)} W_l \text{src}(\hat{u}) dx$$

Et en appliquant la méthode de Galerkin, qui consiste à prendre les fonctions de pondération égale aux fonctions de base :

$$\int_{\Omega(x)} N_k \hat{u}_t dx = \int_{\Omega(x)} N_k \hat{u} dx + \int_{\Omega(x)} N_k \hat{u}_x dx + \int_{\Omega(x)} N_k \hat{u}_{xx} dx + \int_{\Omega(x)} N_k \text{src}(\hat{u}) dx$$

À ce stade il faut donc définir les fonctions  $N_k$ . C'est donc à partir d'ici que la méthode basée sur les éléments lagrangiens et celle basée sur les éléments hermithiens se différencieront.

## 5 Simulateur en éléments finis lagrangiens linéaires.

### 5.1 Rappel théorique

Pour les éléments fini lagrangiens linéaires les fonction de base  $N_k$  sont des fonctions linéaires, différentes de 0 uniquement sur les éléments  $\Omega_k$  et  $\Omega_{k+1}$ . On limitera l'étude sur un seul élément, contenant uniquement 2 noeuds : a et b et sur le quel seulement 2 fonctions de base seront non nulles :  $N_1$  et  $N_2$ . Sur un élément, et en passant en coordonne réduite  $\xi$  on aura donc :

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

$$N_1(\xi) = \frac{1 - \xi}{2} \quad N_2(\xi) = \frac{1 + \xi}{2} \quad \xi = \frac{2}{h_k}(x - x_{ck})$$

où  $h_k$  et  $x_{ck}$  sont la longeur et le centre de l'element  $\Omega_k$ .

Nos équations étant :

$$\frac{\partial v}{\partial t} = D \frac{\partial^2 v}{\partial z^2} + v(v-1)(\mu-v) - w + S$$

$$\frac{\partial w}{\partial t} = \beta_1 v - \beta_2 w$$

et on peut réécrire la première sous la forme :

$$\frac{\partial v}{\partial t} = D \frac{\partial^2 v}{\partial z^2} - \mu v - w + (\mu + 1)v^2 - v^3 + S$$

Nous remarquons que tous les termes de droite de notre première équation ne sont pas linéaire. Cela complexifiera quelque peu la programmation de notre simulateur en éléments finis.

Le traitement des parties linéaires par les éléments finis débouche sur l'établissement des matrices de différentiation, opérateurs analogues aux matrices de dérivation utilisées dans la technique des différences finies.

### 5.2 Modélisation

#### 5.2.1 Main

La partie initiale du programme est tout à fait semblable à celle de la première partie, avec la déclaration des variables globales, initiation des constantes du problème, la création de la grille spatiale et temporelle et l'implémentions des conditions initiales.

```
close all
clear all

% Declaration des variables globales
global D B1 B2 mu k
global z0 zL n dz z h
global xquad wquad
global v0 w0 u0
global D0 D2

% Grille spatiale
z0 = 0;
zL = 50;
n = 201;
nel = n-1;
dz = (zL - z0)/nel;
z = (z0:dz:zL)';
```

```

h= z(2)-z(1);

% constantes du probleme
D = 0.01;
mu = 0.08 ;
k=3;
B1 = 0.008;
B2 = 2.54*B1;
ne=2;

% Conditions initiales
v0 = zeros (1,n);
w0 = zeros (1,n);
u0 =[v0 w0];

%instants de visualisation
dt=0.2;
tmax= 200;
t=0:dt:tmax;
nt=length(t);

```

Entre autre dans la déclaration des constantes du problème on retrouve "ne", le nombre d'équation du système, utilisé plus tard pour la construction de la matrice masse M.

Une fois définie ces généralités on commence à implémenter la méthode des éléments finis.  
On commence par construire les matrices de différentiation :

```

D0=lagrD0_1(h,n);
D0=sparse(D0);
D2=lagrD2_1(h,n);
D2=sparse(D2);

```

Celles ci sont les matrices résultantes du développement des intégrales des termes de droite de l'équation :

$$\int_{\Omega(x)} N_k \hat{u}_t dx = \int_{\Omega(x)} N_k \hat{u} dx + \int_{\Omega(x)} N_k \hat{u}_x dx + \int_{\Omega(x)} N_k \hat{u}_{xx} dx + \int_{\Omega(x)} N_k src(\hat{u}) dx$$

dans la quel chaque intégrale  $\int_{\Omega}$  a été réécrite comme somme d'intégrales sur les éléments fini  $\int_{\Omega_i}$ . On a donc remplacé  $\hat{u}$  par son expression sur un élément en Éléments lagrangiens :

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

et en tenant compte que, sur chaque élément, uniquement 2 fonctions de pondération (égales aux fonctions de base) sont différentes de 0 et des expression des  $N_k$  décrites plus haut, on retrouve les matrices  $M$ ,  $D_0$ ,  $D_1$ ,  $D_2$  développées dans le syllabus relativement correspondantes aux intégrales du terme en  $u_t$ ,  $u$ ,  $u_x$ ,  $u_{xx}$ . Tout calcul fait on peut donc remplacer l'équation précédente par :

$$M \frac{\partial \bar{u}}{\partial t} = \alpha_0 D_0 \bar{u} + \alpha_1 D_1 \bar{u} + \alpha_2 D_2 \bar{u} + \bar{f}_{NL}(\bar{u})$$

où  $\bar{f}_{NL}(\bar{u})$  est la matrice issue du développement de l'intégrale sur le terme non linéaire  $src(\hat{u})$

On voit dans le listing décrit précédemment que seulement  $D_0$  et  $D_2$  sont implémentées. En effet  $D_1$  sera ici nul, vu l'absence de termes en  $v_x$  et en  $w_x$ , alors que la matrice de masse  $M$  et la matrice  $\bar{f}_{NL}(\bar{u})$  seront implémentées plus loin dans le programme.

On trouve également l'utilisation de la fonction *sparse*, qui permet de retenir uniquement les éléments non nuls des matrices, en optimisant le temps de calcul.

Le terme non linéaire  $src(\hat{u})$  va devoir être traité de façon numérique, c'est pourquoi on implémente dans notre programme la formule d'intégration numérique de Gauss à 2 points. Pour ce faire on calcul dans le main les paramètres  $xquad$  et  $wquad$ , qui seront ensuite utilisées utilisés dans le sous-programme Implement.m.

```
nquad = 2;
beta = .5./sqrt(1-(2*(1:nquad)).^(-2));
T=diag(beta,1) + diag(beta,-1);
[V,De]= eig(T);
xquad=diag(De);
[xquad, i] = sort(xquad);
wquad = 2*V(1,i).^2;
```

Finalement on passe à l'intégration temporelle avec l'intégrateur ODE choisi :

```
options= odeset('Mass',masseL1(h,n,ne));
[tout,yout] = ode15s(@Impulse,t,u0,options);
yout = yout (:,1 :length(z));
```

On retrouve ici le calcul de la matrice de masse M au travers de la fonction masseL1. Celle ci est ensuite passé à l'intégrateur en tant qu'option. Comme dans la partie précédente, ode15s fait appel à la fonction Impulse, dans la quel on va devlopper les membres de droite de nos équations. La dernière ligne quant à elle sert à garder dans la solution uniquement les valeurs de v, en éliminant celles de w.

### 5.2.2 Impulse

C'est ici qu'on va "construire" nos équations.

Après avoir déclaré les variables globales on commence par séparer le vecteur d'inconnues  $u$  en 2, à fin de différentier les valeurs de  $v$  et  $w$  et par calculer le terme source  $S$  à l'aide d'une des 3 fonctions source dont on a déjà parlé.

```
v= u(1:n);
w= u(n+1:2*n);
%S=Source(z,t);
%S=Source1(z,t);
S=Source2(z,t);
```

On s'intéresse ensuite à la partie linéaire des équations. Par analogie de notre système avec l'équation :

$$M \frac{\partial \bar{u}}{\partial t} = \alpha_0 D_0 \bar{u} + \alpha_1 D_1 \bar{u} + \alpha_2 D_2 \bar{u} + \bar{f}_{NL}(\bar{u})$$

on obtient :

```
vt= - mu*D0*v - D0 * w + D*D2*v;
wt= B1*D0*v - B2*D0*w;
```

Suit le traitement du terme non linéaire, effectué à l'aide des fonctions Integrant1 et Integrant2, dont le listing se trouve plus loin.

```
y1= feval('integrant1',xquad)*wquad';
y2= feval('integrant2',xquad)*wquad';
```

Dans ces fonctions on développe le terme non linéaire de notre première équation (la deuxième n'en présentant pas) :

$$(\mu + 1)\hat{u}^2 - \hat{u}^3 + S$$

on remplace donc  $\hat{u}$  par

$$u_a(t)N_1(\xi) + u_b(t)N_2(\xi)$$

et on multiplie le résultat par la première fonction de pondération  $N_1$  dans Integrand1 et par  $N_2$  dans Integrand2.

On obtient ainsi les 2 vecteurs  $y1$  et  $y2$  tels que :

$$y1 = \frac{h}{2} \begin{pmatrix} N_1((\mu+1)(u_1N_1 + u_2N_2)^2 - (u_1N_1 + u_2N_2)^3 + S_1) \\ N_1((\mu+1)(u_2N_1 + u_3N_2)^2 - (u_2N_1 + u_3N_2)^3 + S_2) \\ \vdots \\ N_1((\mu+1)(u_{N-1}N_1 + u_NN_2)^2 - (u_{N-1}N_1 + u_NN_2)^3 + S_{N-1}) \end{pmatrix}$$

$$y2 = \frac{h}{2} \begin{pmatrix} N_2((\mu+1)(u_1N_1 + u_2N_2)^2 - (u_1N_1 + u_2N_2)^3 + S_2) \\ N_2((\mu+1)(u_2N_1 + u_3N_2)^2 - (u_2N_1 + u_3N_2)^3 + S_3) \\ \vdots \\ N_2((\mu+1)(u_{N-1}N_1 + u_NN_2)^2 - (u_{N-1}N_1 + u_NN_2)^3 + S_N) \end{pmatrix}$$

On remarque la présence de  $\frac{h}{2}$ , présent à cause du changement de variables.

On rajoute donc ces 2 termes à la première équation.

```
vt(1:n-1)=vt(1:n-1)+y1;
vt(2:n)=vt(2:n)+y2;
```

$y1$  est rajouté aux  $N - 1^{eme}$  premières lignes, alors que  $y2$  aux  $N - 1^{eme}$  dernières.

### 5.2.3 Integrand1

```
u1=v(1:n-1);
u2=v(2:n);

for i=1:length(z)
    [N1, N2] = trialfunctions_lagr_1(z(i));
    u=N1*u1+N2*u2;

    out(1:n-1,i)=(h/2)*N1*((mu+1)*u.^2-u.^3+S(1:n-1));
endfor
```

### 5.2.4 Integrand2

```
u1=v(1:n-1);
u2=v(2:n);

for i=1:length(z)
    [N1, N2] = trialfunctions_lagr_1(z(i));
    u=N1*u1+N2*u2;

    out(1:n-1,i)=(h/2)*N2*((mu+1)*u.^2-u.^3+S(2:n));
endfor
```

### 5.2.5 Résultats

On termine enfin le programme par l'implémentation des conditions aux limites. Dans notre cas elle sont de type Dirichlet, il nous suffit donc de rajouter le terme  $-(u_1 - CL_1(t))$  à la première équation et  $-(u_N - CL_N(t))$  à la dernière.

Pour notre problème on a  $v_0 = v_N = w_0 = w_N = 0$ . Il en suit :

```

vt(1)=vt(1)-v(1);
vt(n)=vt(n)-v(n);
wt(1)=wt(1)-w(1);
wt(n)=wt(n)-w(n);

```

Les 2 vecteurs  $vt$  et  $wt$  sont donc stockés dans un même vecteur  $ut$ , qui sera l'objet de l'intégration temporelle.

Finalement dans le main on affiche la solution, à l'aide de la fonction visualizer. Voici les résultats obtenus pour les 3 sources avec les éléments finis lagrangiens linéaires.

Terme source 1 :

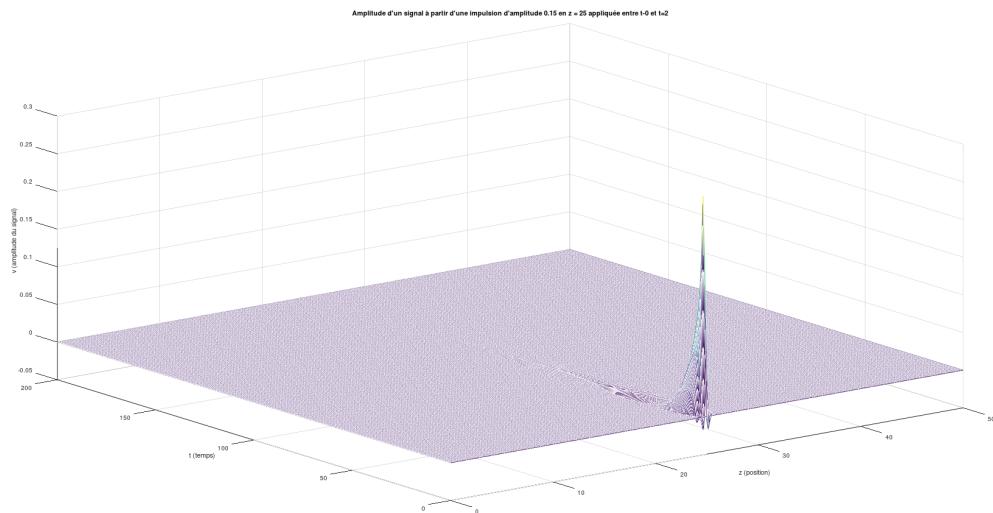


FIGURE 23 – Résultats éléments lagrangiens avec terme source 1 et n=201

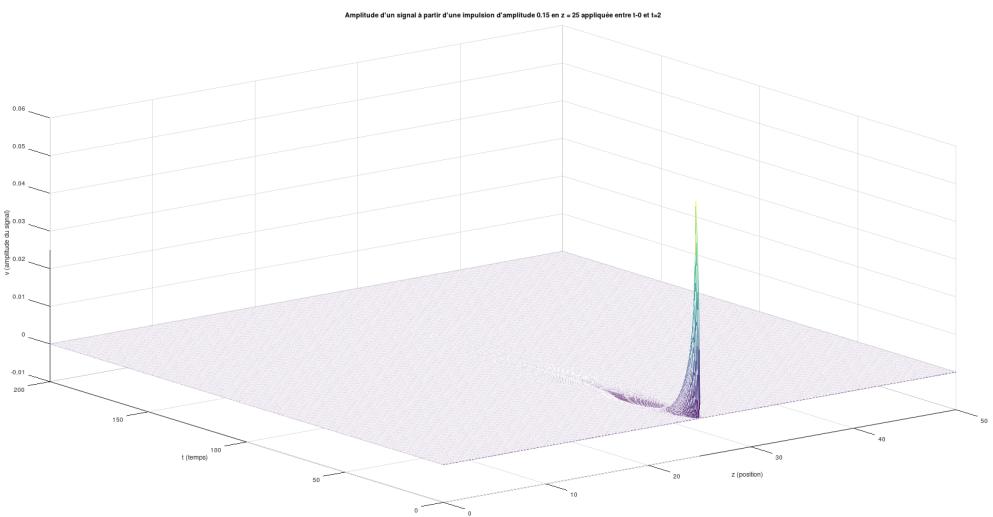


FIGURE 24 – Résultats éléments lagrangiens avec terme source 1 et n=1001

On voit que comme pour la première partie l'amplitude de l'impulsion diminue avec le nombre de points de grille. Ce résultat sont obtenus avec un temps de calcul respectivement de 31,686 s et 67,972 s.

Terme source 2 :

Comme pour les différences finies on voit que le graphique est devenu indépendant du nombre de

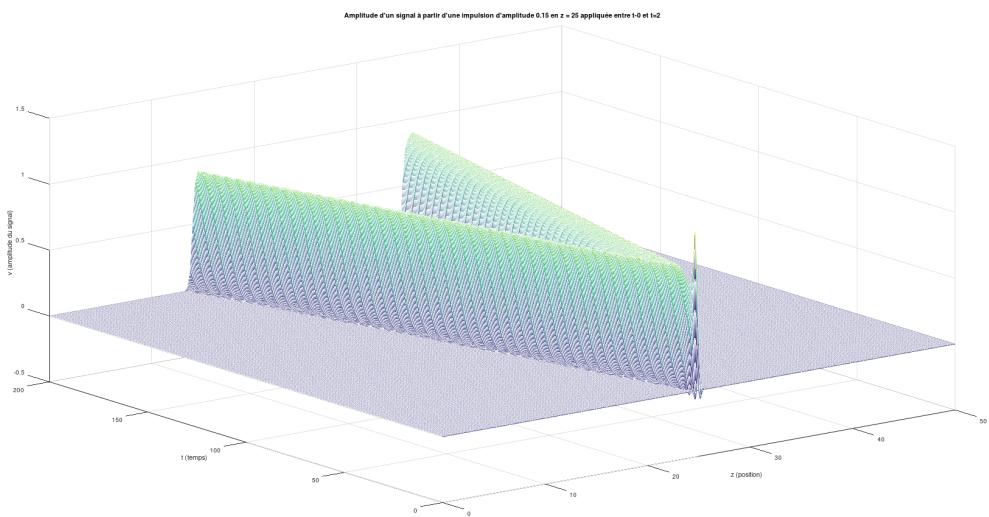


FIGURE 25 – Résultats éléments lagrangiens avec terme source 2 et n=201

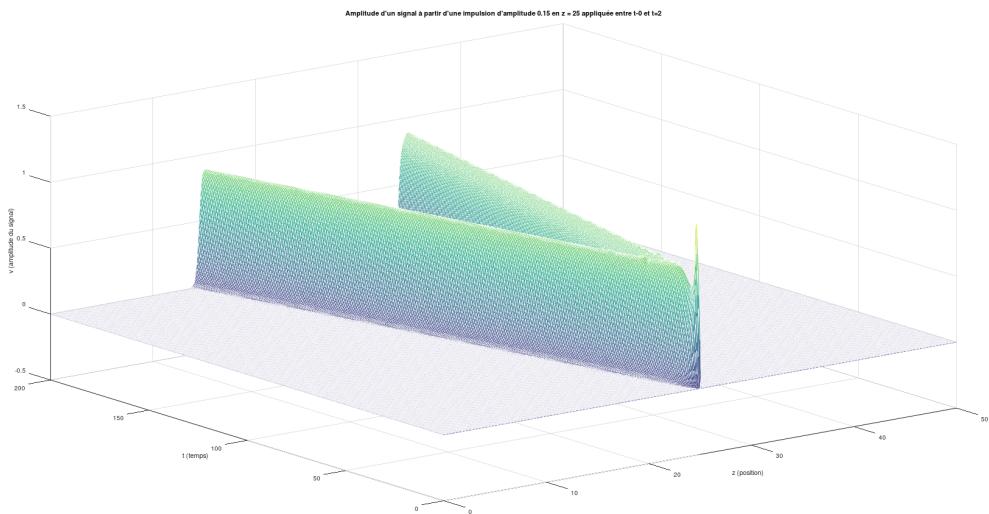


FIGURE 26 – Résultats éléments lagrangiens avec terme source 2 et n=1001

points. Ces résultats sont obtenus avec un temps de calcul respectivement de 67,972 s et 269,51 s.

Terme source 3 (avec k=3) :

Ces résultats sont obtenus avec un temps de calcul respectivement de 28,695 s et 175,07 s.  
Nous analyseront plus en détail ces résultats dans la conclusion.

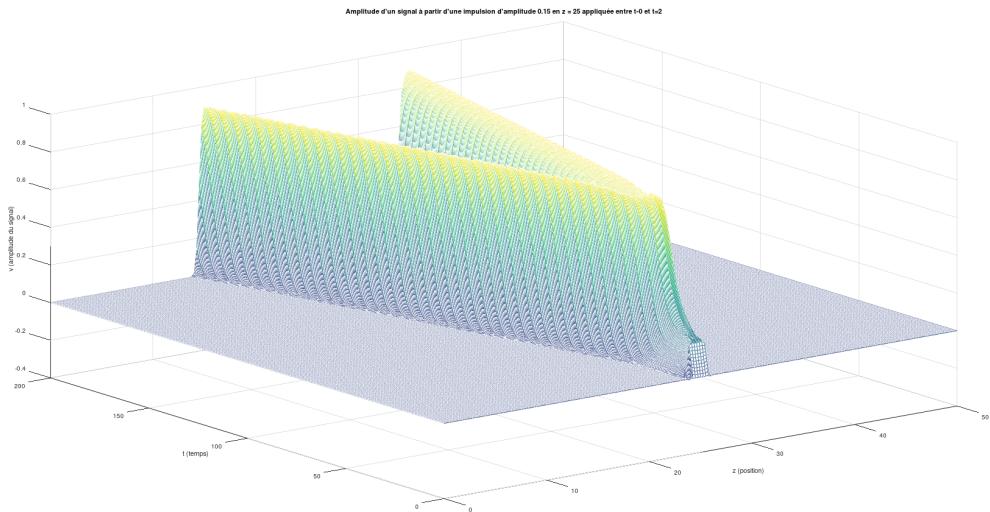


FIGURE 27 – Résultats éléments lagrangiens avec terme source 3 ( $k=3$ ) et  $n=201$

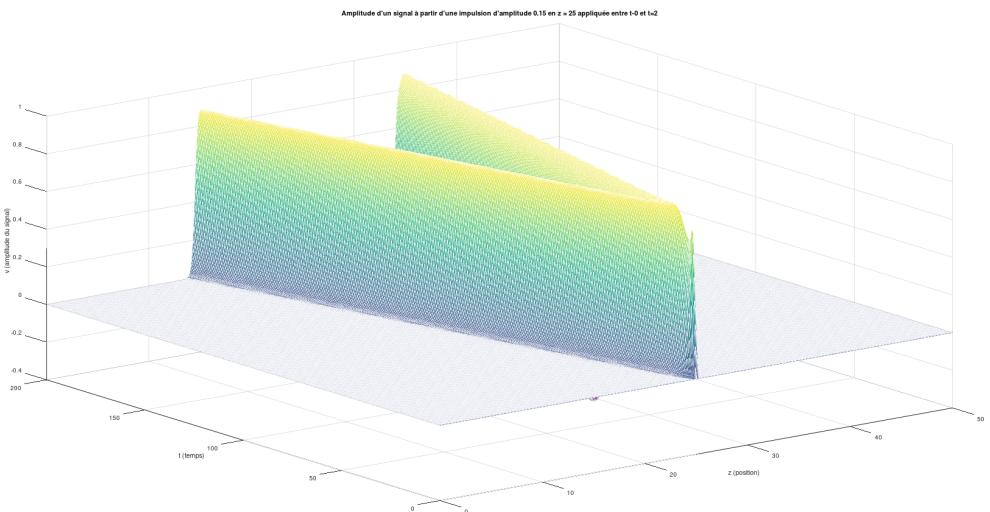


FIGURE 28 – Résultats éléments lagrangiens avec terme source 3 ( $k=3$ ) et  $n=1001$

## 6 Simulateur en éléments finis hermitiens

Dans cette partie, nous allons nous atteler à la réalisation d'un simulateur en éléments finis hermitiens à l'aide de la méthode de Galerkin. Afin de comparer nos résultats avec nos modélisations précédentes, nous utiliserons les mêmes instants de visualisation.

### 6.1 Rappel théorique

Les éléments fini hermitiens se différencient des éléments fini lagrangiens par le choix des fonctions de base  $N_k$ . Ici celles ci sont des fonctions de degrés 3 et, différemment des éléments lagrangiens, les fonctions non nulles sur chaque éléments seront au nombre de 4. Sur un élément  $\Omega_k$  du domaine et en passant en coordonne réduite  $\xi$  on aura donc :

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

$$N_1(\xi) = \frac{1}{4}(1 - \xi)^2(2 + \xi)$$

$$N_2(\xi) = \frac{1}{4}(1 - \xi^2)(1 - \xi)$$

$$N_3(\xi) = \frac{1}{4}(1 + \xi)^2(2 - \xi)$$

$$N_4(\xi) = \frac{1}{4}(-1 + \xi^2)(1 + \xi)$$

On voit aussi une grosse différence dans la formule de l'approximation de  $u$  où, contrairement à celle utilisé en éléments lagrangiens linéaires, sont présentes les dérives de  $u$  par rapport à  $\xi$  :  $u_{a\xi}$  et  $u_{b\xi}$ .

Ce ci impliquera l'augmentation du nombre d'inconnues à calculer d'un facteur 2, tout en ayant toujours seulement 2 noeuds sur chaque élément.

Pour le reste le principe reste le même que celui devloppé dans les elements lagrangiens.

## 6.2 Modélisation

Dans le script Main du dossier sur les éléments hermitiens, nous retrouvons comme dans les autres simulateurs la déclarations des variables globales, la création de la grille spatiale, ainsi que l'initialisation des constantes du problème.

```
close all
clear all

global D B1 B2 mu
global z0 zL n dz z h
global xquad wquad
global v0 w0 u0
global D0 D2
global N1 u S

z0 = 0;
zL = 50;
n = 201;
nel = n-1;
dz = (zL - z0)/nel ;
z = (z0:dz:zL) ';
h= z(2)-z(1);

D = 0.01;
mu = 0.08 ;
B1 = 0.008;
B2 = 2.54*B1;
ne=2;
```

Il s'agit ensuite de calculer les matrices de différentiation, en suivant le même principe que celui pour les éléments lagrangiens.

```
D0=herD0_1(h , n );
D0=sparse(D0);
D2=herD2_1(h , n );
D2=sparse(D2);
```

Nous posons ensuite les conditions initiales. En plus que  $v_0$  et  $w_0$  pour cette partie il est nécessaire de définir également  $v_{0x}$  et  $w_{0x}$ , à cause de l'expression de  $\hat{u}$ . Ces 4 vecteurs de longueur N sont ensuite assemblés dans le vecteur  $u_0$ , qui sera donc de longueur  $4N$ . Le vecteur sera de la forme  $(v_1, v_{x1}, v_2, v_{x2}, \dots, v_N, v_{xN}, w_1, w_{x1}, \dots, w_N, w_{xN})$ .

```
v0 = zeros (1 ,n );
v0x = zeros (1 ,n );
```

```
w0 = zeros (1 ,n) ;
w0x = zeros(1 ,n) ;
u0 (1:2:2*n-1) = v0 ;
u0 (2:2:2*n) = v0x ;
u0(2*n+1:2:4*n-1) = w0;
u0(2*n+2:2:4*n)=w0x ;
```

Nous choisissons nos instants de visualisation avec t comme vecteur de ces différents instants.

```
dt=0.2;
tmax= 200;
t=0:dt:tmax;
nt=length(t);
```

Encore une fois on calcule xquad et wquad pour l'intégration numérique du terme non linéaire.

```
nquad = 2;
beta = .5./sqrt(1-(2*(1:nquad)).^(-2));
T=diag(beta,1) + diag(beta,-1);
[V,De]= eig(T);
xquad=diag(De);
[xquad , i] = sort(xquad);
wquad = 2*V(1,i).^2;
```

Enfin, on lance l'intégration temporelle, on affiche le temps de calcul ainsi que la visualisation graphique du résultat. Avant de visualiser le vecteur solution yout on récupère uniquement une colonne sur 2 à partir de la première jusque la  $2n^{eme}$ , à fin d'obtenir dans la solution uniquement les valeurs de v, et pas celles de  $v_x$ , w et  $w_x$ .

```
tic

options= odeset ('Mass' ,masseH1(h,n,ne));
[tout ,yout] = ode15s (@Impulse ,t ,u0 ,options );
yout = yout (:,1:2:2*length(z));

tcpu=toc;
tcpu

Visualizer(z ,t ,yout);
```

Les résultats obtenus grâce à notre simulateur en éléments finis hermitiens sont très semblables de ceux obtenus avec les 2 autres simulateurs. Voici le résultat pour la simulation effectuée avec le deuxième terme source et un maillage spatiale à 201 points pour comparaison.

Nous pouvons voir que le premier pic est bien présent et visible sur le graphique et que l'onde se propage sur tout le domaine temporel. Nos tests tournent en moyenne autour de 80 à 120 secondes de compilation pour notre simulateur.

La fonction impulse utilisée dans notre intégrateur ode15s commence toujours par la séparation des inconnues v et w, le calcul du terme source S et l'implementation de la partie linéaire de l'équation.

```
v= u(1:2*n);
w= u(2*n+1:4*n);
S1=Source2(z ,t );
S(1:2:2*n-1)=S1;
S(2:2:2*n)=S1;

vt= - mu*D0*v - D0 * w + D*D2*v;
wt= B1*D0*v - B2*D0*w;
```

**Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en z = 25 appliquée entre t=0 et t=2**

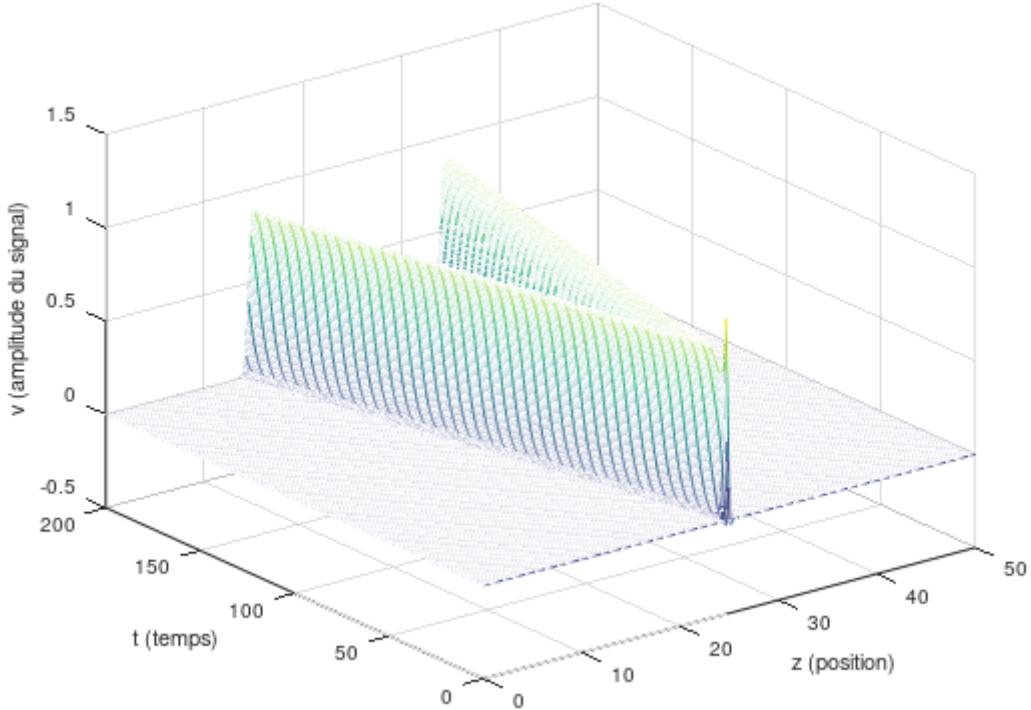


FIGURE 29 – Résultats éléments finis hermitiens deuxième terme source et n=201

tcpu = 110.61

FIGURE 30 – Temps de compilation

Ce pendant l'implementation de S comporte ici une étape supplémentaire due à la présence des termes en dérivé spatial présentes dans la formule de  $\hat{u}$ . On doit en effet dédoubler chaque valeur de S à fin d'obtenir un vecteur de longueur  $2n$ .

Ensuite, la partie non-linéaire est traitée. Différemment des éléments lagrangiens on aura ici 4 fonctions integrand, à cause de la présence de 4 termes dans la formule de  $\hat{u}$ . Chaque fonctions calculera donc le produit d'une des 4 fonctions de pondération (= aux fonctions de base par la méthode de Galerkin) non nulles sur les différents éléments  $\Omega_k$ ,  $N_1$   $N_2$   $N_3$  et  $N_4$ , par  $src(\hat{u})$  qui vaudra ici :

$$src(\hat{u}) = (\mu + 1)(u_a N_1 + u_{a\xi} N_2 + u_b N_3 + u_{b\xi} N_4)^2 - (u_a N_1 + u_{a\xi} N_2 + u_b N_3 + u_{b\xi} N_4)^3 + S$$

On obtient donc les 4 vecteurs  $y1, y2, y3$  et  $y4$ , tous de longueur  $n-1$ . Faut noter que dans  $y2$  et  $y4$  il faudra rajouter la dérivé par rapport à  $\xi$  du terme source aux différents points. On rajoute ensuite ces termes à l'équation, en suivant le schema montré dans la matrice V-50 du syllabus.

```

y1= feval( 'integrand1 ' ,xquad ') *wquad ' ;
y2= feval( 'integrand2 ' ,xquad ') *wquad ' ;
y3= feval( 'integrand3 ' ,xquad ') *wquad ' ;
y4= feval( 'integrand4 ' ,xquad ') *wquad ' ;

vt (1:2:2*n-3)=vt (1:2:2*n-3) + y1;
vt (2:2:2*n-2)=vt (2:2:2*n-2) + y2;
vt (3:2:2*n-1)=vt (3:2:2*n-1) + y3;
vt (4:2:2*n)=vt (4:2:2*n) + y4;

```

Enfin, nous implémentons les conditions aux limites, de la même façon que pour le éléments lagrangiens, et composons le vecteur  $u_t$  qui sera traité dans notre intégrateur.

```

vt(1)=vt(1)-v(1);
vt(2*n-1)=vt(2*n-1)-v(2*n-1);
wt(1)=wt(1)-w(1);
wt(2*n-1)=wt(2*n-1)-w(2*n-1);

ut=[vt; wt];

```

## 7 Comparaison des méthodes

Après avoir implémenté les trois méthodes de résolution d'équation aux dérivées partielles nous allons les comparer entre elles en présentant les différents temps de calcul de chacun des algorithmes pour un problème donné. Ensuite, on observera les différences au niveau de la solution qui est donnée en sortie de chaque simulateur.

### 7.1 Temps de calcul

Nous allons mesurer le temps de calcul pour la résolutions de l'EDP deuxième terme source. Avec 501 points de grille.

Méthode	Différences finie	Lagrange	Hermite
Temps de calcul	1.3421	22.397	130.34

L'avantage du temps de calcul est, évidemment, pour la méthode des différences finies. On observe que la méthode des éléments finis hermitiens prennent un temps de calcul considérablement plus élevé comparé aux deux autres méthodes.

### 7.2 Allure des graphiques

#### 7.2.1 Hermite

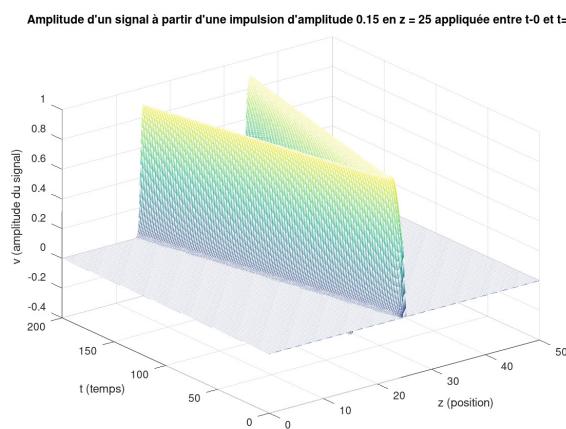


FIGURE 31 – Résultats éléments finis hermitiens deuxième terme source et n=201

## 7.2.2 Lagrange

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

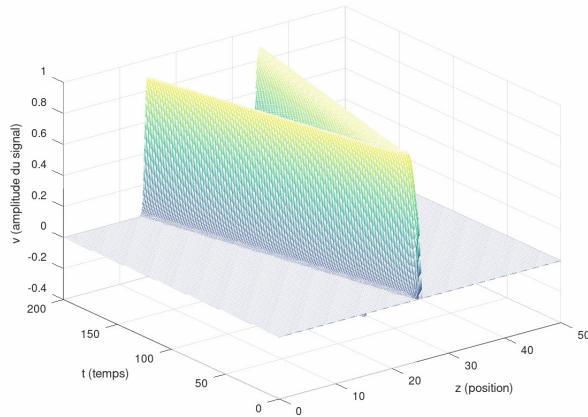


FIGURE 32 – Résultats éléments finis Lagrangien deuxième terme source et  $n=201$

## 7.2.3 Différences finies

Amplitude d'un signal à partir d'une impulsion d'amplitude 0.15 en  $z = 25$  appliquée entre  $t=0$  et  $t=2$

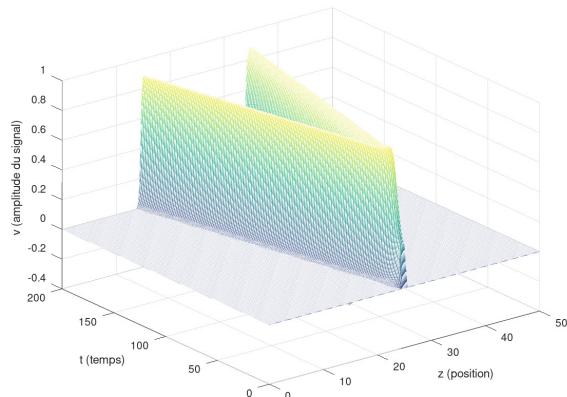


FIGURE 33 – Résultats différences finies deuxième terme source et  $n=201$

On observe que les différences visuelles entre les éléments finis Lagrangiens et les différences finies sont minimes sur la solution de sortie du simulateur, les éléments finis de Lagrange n'apportent, dans ce cas précis, qu'une moindre amélioration (les différences finies font presque aussi bien avec moins de calcul. Les éléments finis hermitiens par contre ont une légère différence par rapport aux deux autres méthodes, ces dernières présentent un pic lisse sans ondulations alors que sur la solution de la méthode hermitienne on observe une légère ondulation sur le pic de la surface.

## Conclusion

Après les études théoriques, nous avons pu implémenter différents algorithmes capables de résoudre notre problème. Nous avons ensuite comparé l'ensemble des méthodes entre elle sur base du temps de calcul et de la qualité de la solution de sortie il en ressort que l'algorithme nécessitant le temps de calcul le plus cours est celui des différences finies suivi de Lagrange et enfin Hermite. Les différences entre chacun de ces algorithmes au niveau de la solution est donc minime voir invisible dans certains cas. Nous avons étudié l'influence du nombre de points ainsi que l'influence du JPattern pour notre problème qui n'apporte rien à la simulation dans nos cas précis.