

# TP Méthodes Numériques pour les EDP : Résolution de l'Équation de la Chaleur par la Méthode des Éléments Finis

ALoui Ghaieth, ROBIN Paul

Mars 2025

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Partie 1 : Résolution théorique</b>	<b>2</b>
2.1	Formulation forte . . . . .	2
2.1.1	Solution Exacte : Cas $f(x) = 1$ . . . . .	2
2.1.2	Solution Exacte : Cas $f(x) = \sin(x)$ . . . . .	4
2.2	Formulation faible . . . . .	5
2.3	Représentation du maillage et des fonctions de base . . . . .	6
2.4	Approximation de la solution . . . . .	7
2.5	Construction de la matrice de rigidité . . . . .	7
2.6	Construction du second membre . . . . .	7
2.6.1	Cas $f(x) = 1$ . . . . .	7
2.6.2	Cas $f(x) = \sin(x)$ . . . . .	7
2.7	Résolution du système linéaire $KU = F$ . . . . .	8
2.8	Solution finale approchée . . . . .	8
2.8.1	Cas $f(x) = 1$ . . . . .	8
2.8.2	Cas $f(x) = \sin(x)$ . . . . .	8
<b>3</b>	<b>Partie 2 : Résolution Numérique</b>	<b>9</b>
<b>4</b>	<b>Validation des fonctions</b>	<b>9</b>

# 1 Introduction

Le but de ce travail est de résoudre l'équation de la chaleur en une dimension par la méthode des éléments finis (EF). Ce problème classique de la physique consiste à résoudre l'équation différentielle :

$$-u''(x) = f(x), \quad 0 < x < L,$$

avec les conditions aux limites de Dirichlet homogènes :

$$u(0) = 0, \quad u(L) = 0.$$

La résolution du problème abordé dans ce travail est structurée en deux grandes parties complémentaires :

La première partie est consacrée à l'analyse théorique du problème, en formulant à la fois la formulation forte et la formulation faible du modèle étudié. Nous y considérons deux cas de source :  $f(x) = 1$  et  $f(x) = \sin(x)$ . Cette partie comprend l'introduction de la formulation variationnelle, la définition de l'espace d'approximation de dimension finie, ainsi que la construction de la matrice de rigidité et du système linéaire associé à la méthode des éléments finis.

La seconde partie porte sur la résolution numérique et l'implémentation du problème en **Julia**. L'objectif est de comparer les solutions obtenues par différentes approches : la solution exacte du problème pour les deux cas de  $f(x)$ , la solution numérique obtenue par l'implémentation du code, ainsi que la solution obtenue par la méthode des éléments finis calculée manuellement. Cette comparaison permettra de valider les résultats numériques et d'évaluer la précision de la méthode.

Enfin, nous analysons l'erreur d'approximation, ainsi que la stabilité et la convergence de la solution numérique. Nous discutons également des améliorations possibles pour optimiser la précision et l'efficacité de la méthode employée.

## 2 Partie 1 : Résolution théorique

### 2.1 Formulation forte

#### 2.1.1 Solution Exacte : Cas $f(x) = 1$

Dans ce cas l'équation différentielle devient :

$$\begin{cases} -u''(x) = 1, & 0 < x < L, \\ u(0) = 0, & \text{(Condition de Dirichlet en } x = 0), \\ u(L) = 0, & \text{(Condition de Dirichlet en } x = L). \end{cases}$$

Soit  $x \in ]0, L[$ , nous intégrons l'équation entre 0 et  $x$  :

$$-\int_0^x u''(t) dt = \int_0^x 1 dt.$$

$$-u'(x) + u'(0) = x.$$

Posons  $C_1 = u'(0)$ , alors :

$$u'(x) = -x + C_1.$$

### Deuxième intégration

Nous intégrons de nouveau :

$$\int_0^x u'(t) dt = \int_0^x (-t + C_1) dt.$$

$$u(x) - u(0) = -\frac{x^2}{2} + C_1x.$$

Posons  $C_2 = u(0)$ , alors :

$$u(x) = -\frac{x^2}{2} + C_1x + C_2.$$

### Détermination des constantes

Les conditions aux limites donnent :

$$u(0) = 0 \Rightarrow C_2 = 0.$$

$$u(L) = 0 \Rightarrow -\frac{L^2}{2} + C_1L = 0.$$

$$C_1 = \frac{L}{2}.$$

### Solution finale

Ainsi, la solution analytique pour  $f(x) = 1$  est :

$$u(x) = -\frac{x^2}{2} + \frac{L}{2}x$$

### 2.1.2 Solution Exacte : Cas $f(x) = \sin(x)$

L'équation devient :

$$-u''(x) = \sin(x), \quad 0 < x < L.$$

#### Première intégration

Nous intégrons entre 0 et  $x$  :

$$-\int_0^x u''(t) dt = \int_0^x \sin(t) dt.$$

$$-u'(x) + u'(0) = -\cos(x) + \cos(0).$$

$$u'(x) = \cos(x) + C_1 - 1.$$

#### Deuxième intégration

Nous intégrons de nouveau :

$$\int_0^x u'(t) dt = \int_0^x (\cos(t) + C_1 - 1) dt.$$

$$u(x) - u(0) = \sin(x) + C_1 x - x.$$

$$u(x) = \sin(x) + C_1 x - x + C_2.$$

#### Détermination des constantes

Les conditions aux limites donnent :

$$u(0) = 0 \Rightarrow 0 + C_2 = 0 \Rightarrow C_2 = 0.$$

$$u(L) = 0 \Rightarrow \sin(L) + C_1 L - L = 0.$$

$$C_1 = \frac{L - \sin(L)}{L}.$$

#### Solution finale

Ainsi, la solution analytique pour  $f(x) = \sin(x)$  est :

$$\boxed{u(x) = \sin(x) + \frac{-\sin(L)}{L}x}$$

## 2.2 Formulation faible

On cherche une solution dans l'espace fonctionnel :

$$V = H_0^1(0, L) = \{v \in H^1(0, L) \mid v(0) = v(L) = 0\}. \quad (1)$$

On multiplie l'équation différentielle par une fonction test  $v \in V$  et on intègre sur  $\Omega$  :

$$\int_0^L -u''(x)v(x) dx = \int_0^L f(x)v(x) dx. \quad (2)$$

En intégrant par parties et en utilisant les conditions aux limites homogènes, on obtient :

$$\int_0^L u'(x)v'(x) dx = \int_0^L f(x)v(x) dx. \quad (3)$$

La formulation faible consiste alors à trouver  $u \in V$  tel que :

$$a(u, v) = l(v), \quad \forall v \in V, \quad (4)$$

où :

- La **forme bilinéaire** est définie par :

$$a(u, v) = \int_0^L u'(x)v'(x) dx. \quad (5)$$

- La **forme linéaire** est donnée par :

$$l(v) = \int_0^L f(x)v(x) dx. \quad (6)$$

### Existence et Unicité de la Solution

Le théorème de Lax-Milgram garantit qu'il existe une unique solution  $u \in V$  si la forme bilinéaire  $a(., .)$  satisfait les conditions suivantes :

- **Continuité** Il existe une constante  $C > 0$  telle que :

$$|a(u, v)| \leq C \|u\|_{H^1} \|v\|_{H^1}, \quad \forall u, v \in V. \quad (7)$$

- **Coercivité** Il existe une constante  $\alpha > 0$  telle que :

$$a(v, v) \geq \alpha \|v\|_{H^1}^2, \quad \forall v \in V. \quad (8)$$

Nous allons maintenant approximer la solution  $u(x)$  en utilisant la méthode des éléments finis avec des éléments  $P_1$  linéaires sur un maillage avec deux éléments définis sur  $[0, L]$ .

## Discrétisation du domaine

Afin de simplifier le calcul à la main nous allons diviser le domaine  $[0, L]$  en **deux éléments** égaux avec trois nœuds :

$$x_0 = 0, \quad x_1 = \frac{L}{2}, \quad x_2 = L.$$

On utilise des **éléments finis**  $P_1$  **linéaires**, définis localement sur chaque sous-intervalle.

- $\varphi_0(x)$  est la fonction de base associée à  $x_0$ ,
- $\varphi_1(x)$  est la fonction de base associée à  $x_1$ ,
- $\varphi_2(x)$  est la fonction de base associée à  $x_2$ .

Elles sont définies comme suit :

$$\varphi_0(x) = \begin{cases} 1 - \frac{2x}{L}, & 0 \leq x \leq L/2, \\ 0, & L/2 < x \leq L. \end{cases}$$

$$\varphi_1(x) = \begin{cases} \frac{2x}{L}, & 0 \leq x \leq L/2, \\ 2 - \frac{2x}{L}, & L/2 < x \leq L. \end{cases}$$

$$\varphi_2(x) = \begin{cases} 0, & 0 \leq x \leq L/2, \\ \frac{2x}{L} - 1, & L/2 < x \leq L. \end{cases}$$

### 2.3 Représentation du maillage et des fonctions de base

Le graphe ci-dessous représente le maillage et les fonctions de base  $P_1$  :

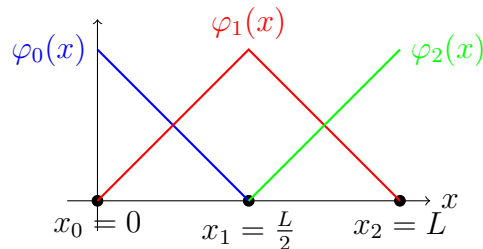


Figure 1: Maillage et fonctions de base  $P_1$

## 2.4 Approximation de la solution

On approxime la solution sous la forme :

$$u_h(x) = U_0\varphi_0(x) + U_1\varphi_1(x) + U_2\varphi_2(x).$$

Les conditions de Dirichlet imposent :

$$U_0 = 0, \quad U_2 = 0.$$

Nous devons donc déterminer uniquement  $U_1$ .

## 2.5 Construction de la matrice de rigidité

La forme bilinéaire est :

$$a(u, v) = \int_0^L u'(x)v'(x) dx.$$

En calculant les dérivées des fonctions de base et en intégrant, on obtient la **matrice de rigidité globale** :

$$K = \frac{2}{L} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

## 2.6 Construction du second membre

La forme linéaire est donnée par :

$$F_i = \int_0^L f(x)\varphi_i(x) dx, \quad i = 0, 1, 2.$$

### 2.6.1 Cas $f(x) = 1$

Nous avons déjà obtenu le vecteur du second membre dans ce cas :

$$F = \frac{L}{4} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}.$$

### 2.6.2 Cas $f(x) = \sin(x)$

Nous devons calculer les termes  $F_i = \int_0^L \sin(x)\varphi_i(x) dx$ , pour  $i = 0, 1, 2$ . Les fonctions de base  $\varphi_i(x)$  sont linéaires et définies par morceaux sur chaque élément.

## 2.7 Résolution du système linéaire $KU = F$

Le système à résoudre est :

$$\frac{2}{L} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} F_0 \\ F_1 \\ F_2 \end{bmatrix}.$$

En résolvant ce système, nous obtenons les valeurs  $U_0$ ,  $U_1$  et  $U_2$ . Comme  $U_0 = U_2 = 0$  (conditions aux limites), nous avons  $U_1$  qui est donné par la solution de l'équation réduite.

## 2.8 Solution finale approchée

### 2.8.1 Cas $f(x) = 1$

Dans ce cas, la solution approchée est donnée par :

$$u_h(x) = \frac{L^2}{8} \varphi_1(x) \Rightarrow u_h(x) = \begin{cases} \frac{L^2}{8} \cdot \frac{2}{L} x = \frac{L}{4} x, & 0 \leq x \leq L/2, \\ \frac{L^2}{8} \left(2 - \frac{2}{L} x\right) = \frac{L^2}{4} - \frac{L}{4} x, & L/2 \leq x \leq L. \end{cases}$$

### 2.8.2 Cas $f(x) = \sin(x)$

La solution approchée s'écrit sous la même forme avec les valeurs obtenues pour  $U_1$  et  $U_2$ . On a donc :

$$u_h(x) = U_1 \varphi_1(x)$$

Les valeurs  $U_1$  est obtenue en résolvant le système linéaire  $KU = F$ .



## 3 Partie 2 : Résolution Numérique

Dans la première partie, nous avons observé les difficultés liées au calcul par la méthode des éléments finis, notamment lorsqu'un grand nombre d'éléments est impliqué. De plus, il est évident que trouver une solution exacte au problème devient complexe lorsque la fonction  $f$  est difficile à intégrer. C'est pourquoi il est nécessaire de résoudre ce problème de manière numérique. Nous allons suivre une série d'étapes, implémentées en **Julia**, pour obtenir une solution complète.

La première partie jouera un rôle crucial pour vérifier le bon fonctionnement et la validation de nos fonctions.

## 4 Validation des fonctions

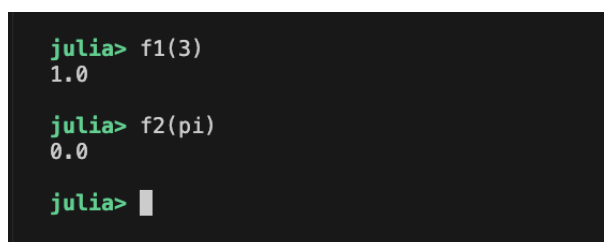
Pour la validation de chaque fonction, nous allons créer un fichier `validation.jl` dans lequel nous placerons nos fonctions. Ensuite, dans un terminal Julia, nous chargerons ce fichier en exécutant la commande suivante :

```
include("validation.jl")
```

Une fois le fichier chargé, nous pourrions tester nos fonctions directement dans le terminal en appelant leur nom. Nous observerons ainsi la sortie de chaque fonction en fonction de l'entrée fournie.

### Étape 1 : Validation des fonctions `f1` et `f2`

Les fonctions `f1` et `f2` sont des fonctions simples qui prennent un nombre réel en entrée et retournent des valeurs prédéfinies ou basées sur des expressions mathématiques simples.



```
julia> f1(3)
1.0

julia> f2(pi)
0.0

julia> █
```

Figure 2: Les résultats obtenus avec les fonctions `f1` et `f2`.

Les résultats sont corrects et conformes aux attentes.

## Étape 2 : Validation de la fonction maillage

La fonction `maillage` génère un maillage 1D avec  $NE$  éléments et  $NN = NE + 1$  noeuds, et l'écrit dans un fichier spécifié.

### Validation :

Nous avons testé cette fonction avec une taille  $L = 2$  et un nombre d'éléments  $NE = 2$ . Le fichier génère un fichier `maillage.txt` qui a été inspecté pour vérifier les éléments suivants :

```
julia> maillage(2,2,"maillage.txt")
julia> 
```

```
≡ maillage.txt
1  3 2
2  1 0.0
3  2 1.0
4  3 2.0
5  1 1 2
6  2 2 3
7  2
8  1
9  3
10
```

Figure 3: Les résultats de la fonction maillage

## Étape 3 : Validation de la fonction lecture\_maillage

La fonction `lecture_maillage` lit les données du fichier de maillage et renvoie les informations sur les noeuds, les éléments et les conditions de Dirichlet.

### Validation :

Nous avons comparé les données extraites avec celles qui étaient écrites dans le fichier "maillage.txt". Les résultats étaient corrects :

```
julia> lecture_maillage("maillage.txt")
(Any[0.0, 1.0, 2.0], Any[1 2; 2 3], Any[1, 3])
julia> 
```

Figure 4: Les résultats de la fonction lecture maillage

## Étape 4 : Validation de la fonction `tables_elementaires`

La fonction `tables_elementaires` calcule la matrice de rigidité  $Ke$  et le second membre  $Fe$  pour un élément donné.

### Validation :

Les valeurs de  $Ke$  et  $Fe$  pour l'élément 1 ont été vérifiées. Nous avons comparé les résultats calculés avec les attentes théoriques pour des éléments finis sur un problème simple (comme  $f(x) = 1$ ). Les résultats étaient conformes aux attentes :

```
julia> liste_noeuds,liste_elements,noeud_dirichlet=lecture_maillage("maillage.txt")
(Any[0.0, 1.0, 2.0], Any[1 2; 2 3], Any[1, 3])

julia> tables_elementaires(2,liste_noeuds,liste_elements,f1)
([1.0 -1.0; -1.0 1.0], [0.5, 0.5])

julia> █
```

Figure 5: Les résultats de la fonction `tables élémentaires`

Les résultats sont valides et correspondent à ce que l'on attend dans la méthode des éléments finis.

## Étape 5 : Validation de la fonction `assembler_matrices`

La fonction `assembler_matrices` assemble les matrices élémentaires en une matrice globale de rigidité  $K$  et un vecteur du second membre  $F$ .

### Validation :

Les matrices  $A$  et  $b$  obtenues ont été vérifiées pour un petit maillage. Les résultats étaient cohérents avec les matrices de rigidité globales attendues pour  $L = 2$  et deux éléments:

```
julia> assembler_matrices(liste_noeuds,liste_elements,f1)
([1.0 -1.0 0.0; -1.0 2.0 -1.0; 0.0 -1.0 1.0], [0.5, 1.0, 0.5])

julia> █
```

Figure 6: Les résultats de la fonction `assemblermatrices`

Les résultats sont validés et la fonction semble fonctionner correctement.

## Étape 6 : Validation de la fonction `imposer_cl_dirichlet`

La fonction `imposer_cl_dirichlet` applique les conditions aux bords de Dirichlet en annulant les lignes et colonnes correspondantes dans la matrice globale et en imposant les valeurs de Dirichlet dans le vecteur du second membre.

### Validation :

La matrice  $A$  et le vecteur  $b$  ont été modifiés comme attendu :

```
julia> A_g,b_g=assembler_matrices(liste_noeuds,liste_elements,f1)
([1.0 -1.0 0.0; -1.0 2.0 -1.0; 0.0 -1.0 1.0], [0.5, 1.0, 0.5])

julia> imposer_cl_dirichlet(A_g,b_g,noeud_dirichlet)
([1.0 0.0 0.0; 0.0 2.0 0.0; 0.0 0.0 1.0], [0.0, 1.0, 0.0])

julia> █
```

Figure 7: Les résultats de la fonction `imposer_cl_dirichlet`

Les résultats sont validés.

## Étape 7 : Validation de la fonction `resoudre_systeme`

La fonction `resoudre_systeme` résout le système d'équations linéaires à l'aide de méthodes appropriées (Cholesky).

### Validation :

Nous avons testé la fonction avec un système de petite taille et vérifié que la solution obtenue était correcte.

Afin de valider la fonction `resoudre_systeme`, nous avons effectué des tests sur deux systèmes : un petit système  $3 \times 3$  et un grand système  $10000 \times 10000$ .

### Test sur un petit système ( $3 \times 3$ )

Nous avons créé une matrice  $A_{\text{small}}$  de taille  $3 \times 3$  et un vecteur  $b_{\text{small}}$  de taille 3. La fonction `resoudre_systeme` devrait utiliser la méthode  $LU$  pour résoudre ce système, étant donné que la taille de la matrice est inférieure à 5000.

Le système testé est :

$$A_{\text{petit}} = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 3 \end{bmatrix}, \quad b_{\text{petit}} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

La solution du système  $A_{\text{small}} \times x = b_{\text{small}}$  a été calculée à l'aide de la fonction `resoudre_systeme`. La vérification de la solution consiste à s'assurer que  $A_{\text{small}} \times x = b_{\text{small}}$ .

## Test sur un grand système ( $10000 \times 10000$ )

Pour tester un grand système, nous avons créé une matrice  $A_{\text{large}}$  de taille  $10000 \times 10000$  en générant une matrice aléatoire symétrique définie positive (en ajoutant l'identité à une matrice aléatoire), et un vecteur  $b_{\text{large}}$  de taille 10000.

Pour obtenir une matrice symétrique définie positive, nous avons symétrisé la matrice  $A_{\text{large}}$  en effectuant  $A = A + A^T$ , puis nous avons ajouté l'identité afin de garantir qu'elle soit définie positive.

Le système testé est donc un système de grande taille  $A_{\text{large}} \times x = b_{\text{large}}$ , où  $A_{\text{large}}$  est une matrice  $10000 \times 10000$ , et  $b_{\text{large}}$  est un vecteur de taille 10000.

Dans ce cas, la méthode Cholesky doit être utilisée pour résoudre le système, car la taille de la matrice est supérieure à 5000.

La solution a été calculée à l'aide de la fonction `resoudre_systeme`, et la vérification de la solution consiste à vérifier que  $A_{\text{large}} \times x \approx b_{\text{large}}$ .

## Vérification de la solution

La vérification des solutions pour les deux systèmes a été effectuée en vérifiant si :

$$A \times x = b$$

```
julia> x_small = resoudre_systeme(A_small, b_small)
3-element Vector{Float64}:
 0.0
 0.0
 1.0

julia> println("Solution pour le système 3x3 : ", x_small)

# Vérification de la solution pour le petit système
Solution pour le système 3x3 : [0.0, 0.0, 1.0]

julia> println("Vérification pour le petit système : ", A_small * x_small == b_small)
Vérification pour le petit système : true
```

Figure 8: Les résultats de la fonction `resoudre_systeme`

## Étape 8 : Validation de la fonction affichage

La fonction `affichage` génère un graphique comparant la solution exacte et la solution numérique obtenue par éléments finis.

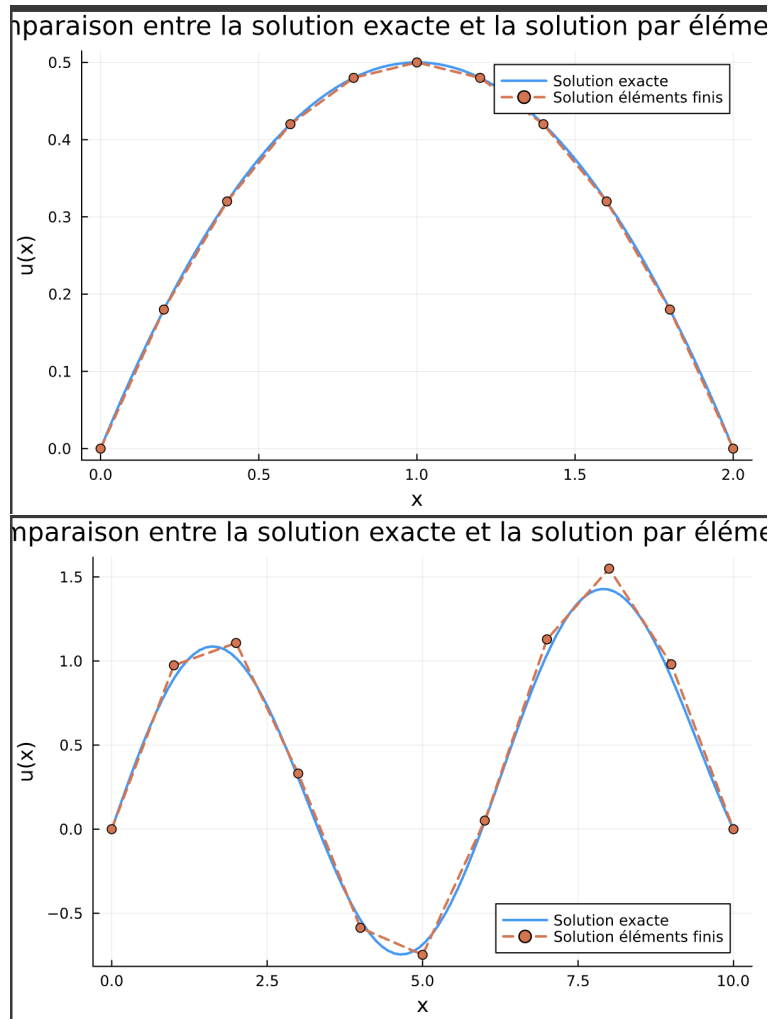


Figure 9: Les résultats de la fonction affichage

Nous avons effectué des tests avec la fonction  $f = 1$  pour 10 éléments et  $L = 2$ , ainsi qu'avec la fonction  $f = \sin$  pour 10 éléments et  $L = 10$ .

### Validation :

Les graphiques générés ont été inspectés visuellement. La solution numérique converge bien vers la solution exacte, ce qui est conforme à ce qui est attendu.

## Étape 9 : Validation de la fonction `calcul_erreur`

La fonction `calcul_erreur` calcule l'erreur entre la solution numérique et la solution exacte pour différentes tailles de maillage.

### Validation :

Les erreurs ont été calculées et tracées en fonction de la taille du maillage. Les résultats montrent une convergence des erreurs comme prévu pour les méthodes des éléments finis.

## Étape 10 : Validation de la fonction `imposition_cl_lim_alternative`

La fonction `imposition_cl_lim_alternative` applique une méthode alternative pour imposer les conditions de Dirichlet et réintégrer la solution dans le vecteur final.

### Validation :

Les résultats ont été vérifiés en comparant les matrices et vecteurs après l'application des conditions de Dirichlet. La solution finale après réintégration était correcte, comme attendu.

```
julia> affichage(A_g,b_g,noeud_dirichlet)
Matrice après suppression des lignes et des colonnes pour cl Dirichlet:
[2.0;;]
Vecteur du second membre après la modification:
[1.0]
La solution complète après la réintégration:
[0.0]
julia> 
```

Figure 10: Les résultats de la fonction `affichage`

Les tests ont validé la fonction.

## Validation des étapes 11, 12

La tâche consiste à modifier la fonction d'écriture du maillage de l'étape 2 afin de produire un fichier de maillage 2D pour un carré  $[0, L] \times [0, L]$ . La structure du maillage a été inspirée du TD 8, où les points sont disposés selon une grille régulière dans le domaine défini. Les étapes suivantes ont été suivies pour cette validation :

1. **Vérification** : La lecture du fichier a été testée avec plusieurs fichiers de maillage 2D pour vérifier la robustesse de la fonction. Les noeuds et les éléments ont été extraits et affichés correctement.

```
1 9 8
2 1 0.0 0.0
3 2 0.5 0.0
4 3 1.0 0.0
5 4 0.0 0.5
6 5 0.5 0.5
7 6 1.0 0.5
8 7 0.0 1.0
9 8 0.5 1.0
0 9 1.0 1.0
1 1 1 2 4
2 2 2 5 4
3 3 2 3 5
4 4 3 6 5
5 5 4 5 7
6 6 5 8 7
7 7 5 6 8
8 8 6 9 8

julia> maillage_2d(1.0,2,"maillage2D.txt")
julia> 
```

Figure 11: Les résultats de la fonction `maillage2D`