2019/2020

Master 2 Recherche fondamentale en mathématiques

Méthode des éléments finis

Professeur : Nicolas Seguin

Rédigé par Pierre Le Barbenchon

Table des matières

1 Introduction

On veut étudier la méthode des éléments finis en dimension 1 pour l'équation

$$\begin{cases} -u'' = f & \text{sur } [0, 10] \\ u(0) = u(10) = 0 \end{cases}$$

On commencera, en partie 2, par des aspects théoriques sur la méthode des éléments finis qui motiveront cette étude.

Ensuite, on traitera dans deux grandes parties 3 et 4, les méthodes liées aux polynômes de Lagrange de degrés 1 et 2 sur des maillages uniforme et non uniforme. L'exemple de maillage non uniforme qu'on donnera est donné par

```
[x*b/(n+1)* sinh(x/n)/sinh((n+1)/n) for x in range (n+2)]
```

Dans chacune des deux parties 3 et 4, on aura une implémentation de la méthode avec, pour fonction f, la fonction $x \mapsto \sin(\pi x)$ sur $[0, 10]$. Puis pour ce problème, on étudiera l'ordre de convergences des méthodes pour les normes $\|\cdot\|_{L^2}$ et $\|\cdot\|_{H^1}$ (que l'on abrègera en norme L^2 et norme H^1) pour les deux maillages (uniforme et non uniforme).

Enfin, en annexe 5, on donnera le code qui permet d'effectuer les simulations, ainsi que la preuve du théorème 2.

L'implémentation se fera dans le langage Python, c'est pourquoi les indices du code commencent à 0.

La plupart des résultats que l'on présente ici sont des restrictions de théorèmes plus généraux (par exemple, le lemme de Céa, les majorations d'erreur), mon objectif étant de bien comprendre les résultats dans notre cadre d'application (Lagrange 1 et 2, pour les normes L^2 et H^1 avec des maillages uniforme et non uniforme). On présente en annexe l'énoncé général de certains résultats (voir 5.6).

2 Motivation et prérequis

Soit $I = [0, 10]$ et l'équation

$$\begin{cases} -u'' = f & \text{sur } I \\ u(0) = u(10) = 0 \end{cases} \tag{1}$$

On pose une formule variationnelle associée au problème (1) :

$$\begin{cases} \text{Trouver } u \in H_0^1(I) \text{ tel que} \\ \forall v \in H_0^1(I), \quad a(u, v) = b(v) \end{cases} \tag{2}$$

où $a(u, v) = \displaystyle\int_I u'v'$ et $b(v) = \displaystyle\int_I fv$.

On veut utiliser une méthode de Galerkin, en se donnant un sous espace de dimension finie V_h (pour $h > 0$) de $V = H_0^1(I)$ qui tend vers V quand h tend vers 0 dans un certain sens.

On veut donc résoudre le problème suivant :

$$\begin{cases} \text{Trouver } u_h \in V_h \text{ tel que} \\ \forall v_h \in V_h, \quad a(u_h, v_h) = b(v_h) \end{cases} \tag{3}$$

Soit N la dimension de l'espace V_h, on se donne une base $(\phi_i)_{i=1}^N$ de l'espace V_h.

Pour résoudre l'équation (3), on procède par analyse-synthèse. Pour l'analyse, on prend une solution u_h de (3), on peut alors écrire

$$u_h = \sum_{j=1}^{N} u_j \phi_j$$

pour $(u_j)_{j=1}^N \in \mathbb{R}^N$.

On a donc $a(u_h, \phi_i) = \displaystyle\sum_{j=1}^{N} u_j a(\phi_j, \phi_i) = b(\phi_i)$ pour tout $i \in \{1, \dots, N\}$, ce qui peut se réécrire en

$$A_h U_h = F_h$$

avec $(A_h)_{i,j} = a(\phi_j, \phi_i)$, $(U_h)_i = u_i$ et $(F_h)_i = b(\phi_i)$.

Dans notre cas, pour résoudre (1), on a

$$(A_h)_{i,j} = \int_I \phi_i'(x)\phi_j'(x)dx$$
$$(F_h)_i = \int_I f(x)\phi_i(x)dx$$

Pour la synthèse, en calculant $U_h = A_h^{-1}F_h$, on trouve aussi que u_h, associé à U_h, est bien solution de (3).

Tout le but des parties suivantes sera de résoudre cette équation avec des V_h différents (et donc des bases (ϕ_i) différentes) et d'étudier les propriétés de convergence entre la solution u_h de (3) et la solution u de (2) quand h tend vers 0.

3 Méthode de Lagrange pour des polynômes de degré au plus 1

3.1 Théorie

Soit I l'intervalle $[0, 10]$.

Définition 1. Soit $N \in \mathbb{N}$. On définit un maillage de I relatif aux N points distincts $(x_i)_{i=1}^N \in]0, 10[^N$ par l'ensemble

$$\tau_h = \{[x_i, x_{i+1}], \quad \forall i \in \{0, \cdots, N\}\},$$

où $x_0 = 0$ et $x_{N+1} = 10$.

Remarque :
On dira qu'un maillage est uniforme si les points $(x_i)_{i=0}^{N+1}$ sont uniformément répartis dans I.

Définition 2. Soit τ_h un maillage, on appelle cellule un élément de τ_h.

On se fixe un maillage τ_h de I.

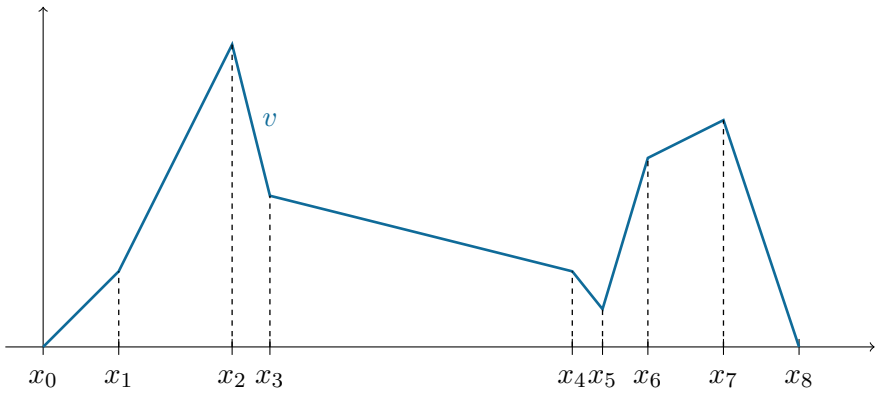
Définition 3. L'espace d'approximation $V_{h,1}$ que l'on va utiliser ici sera

$$V_{h,1} = \left\{ v \in \mathcal{C}^0(I), \quad v_{|[x_j, x_{j+1}]} \in \mathbb{P}_1[X], \quad j = 0, \ldots, N \text{ avec } v(0) = v(10) = 0 \right\}$$

où $\mathbb{P}_1[X]$ désigne les polynômes réels de degré au plus 1.

Proposition 4. L'espace $V_{h,1}$ est de dimension N.

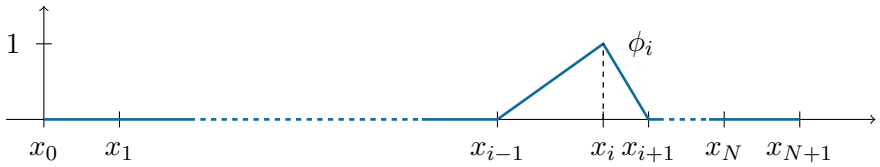
Exemple 5. La fonction v dessinée ci-dessous est dans $V_{h,1}$. (pour $N = 7$)



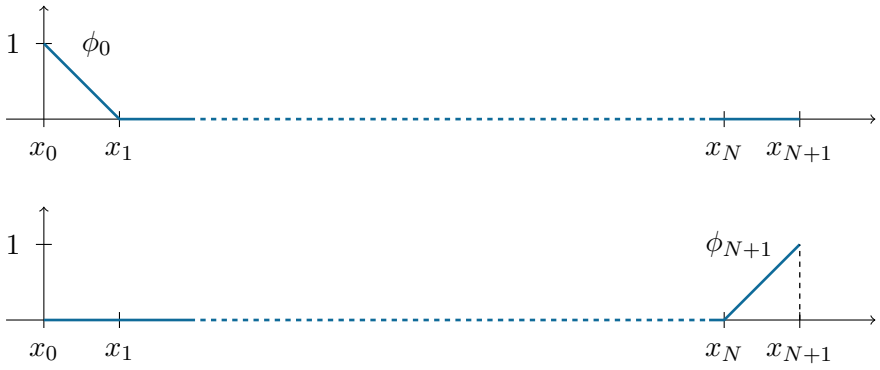
Il faut se donner une base de $V_{h,1}$.

Définition 6. On prend la base $\mathcal{B} = \{\phi_i \in V_{h,1}, i \in \{1, \dots, N\},$ où $\phi_i(x_j) = \delta_{i,j}\}$.

Les fonctions ϕ_i sont de la forme suivante



On pose les fonctions ϕ_0 et ϕ_{N+1} de la même façon.

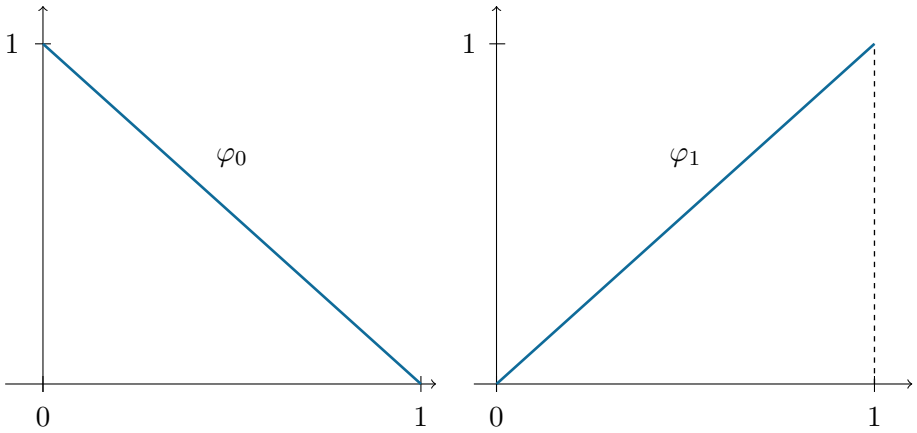


Comme vu dans la partie 2, on pose alors les matrices

$$(A_h)_{i,j} = \int_I \phi_i'(x)\phi_j'(x)dx \text{ pour tout } i,j \in \{1,\dots,N\}$$
$$(F_h)_i = \int_I f(x)\phi_i(x)dx \text{ pour tout } i \in \{1,\dots,N\}$$

La solution est alors $U_h = A_h^{-1}F_h$.

Pour calculer les matrices A_h et F_h de manière pratique, on se donner une cellule de référence $\hat{K} = [0,1]$ sur laquelle on se donne deux fonctions de base φ_0 et φ_1.



Ainsi, pour une cellule $K = [\alpha, \beta]$ de τ, on peut définir la transformation affine

$$F_K : \begin{cases} \hat{K} & \to & K \\ \hat{x} & \mapsto & \hat{x}\beta + (1-\hat{x})\alpha \end{cases} .$$

Cette transformation envoie $\hat{K}$ sur K.
Ainsi, pour $K = [x_j, x_{j+1}]$, on note $\phi_{K,0} = \phi_j$ et $\phi_{K,1} = \phi_{j+1}$, on a alors, pour $l, m \in \{0, 1\}$,

$$\begin{aligned}
\int_K \phi'_{K,l}(x)\phi'_{K,m}(x)dx &= \int_K (\varphi_l \circ F_K^{-1})'(x)(\varphi_m \circ F_K^{-1})'(x)dx \\
&= \int_K (F_K^{-1})'(x)\varphi'_l(F_K^{-1}(x))(F_K^{-1})'(x)\varphi'_m(F_K^{-1}(x))dx \\
&= \int_K \frac{1}{F'_K(F_K^{-1}(x))}\varphi'_l(F_K^{-1}(x))\frac{1}{F'_K(F_K^{-1}(x))}\varphi'_m(F_K^{-1}(x))dx \\
&= \frac{1}{(x_{j+1}-x_j)^2}\int_K \varphi'_l(F_K^{-1}(x))\varphi'_m(F_K^{-1}(x))dx \\
&= \frac{1}{(x_{j+1}-x_j)^2}\int_{\hat{K}} \varphi'_l(\hat{x})\varphi'_m(\hat{x})(x_{j+1}-x_j)d\hat{x} \\
&= \frac{1}{x_{j+1}-x_j}\int_{\hat{K}} \varphi'_l(\hat{x})\varphi'_m(\hat{x})d\hat{x}
\end{aligned}$$

On va donc pouvoir se servir de cette formule sur la cellule de référence pour calculer toutes les intégrales de la matrices A_h. De même, on peut faire le même type de calcul pour calculer F_h.

3.2 Implémentation numérique

On veut résoudre l'équation

$$\begin{cases} -u''(x) = \sin(\pi x) \quad \text{sur } [0, 10] \\ u(0) = u(10) = 0 \end{cases} \tag{4}$$

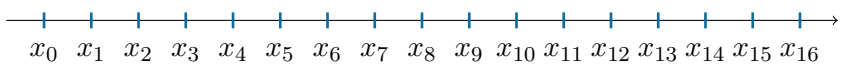
La solution exacte est

$$\forall x \in [0, 10], \quad u(x) = \frac{\sin(\pi x)}{\pi^2}$$

J'ai choisi d'implémenter le maillage par une liste qui contient les $N+2$ points rangés par ordre croissant (donc les 2 bords sont aux extrémités de la liste).

Pour un maillage uniforme, on aura

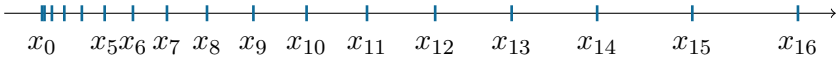
```
Nodes_1_uni = [ b*i/(n+1) + (1-i/(n+1))*a for i in range(n+2)]
```



Exemple pour $N = 15$

On peut se donner un maillage non uniforme comme

```
Nodes_1 = [x*b/(n+1)* sinh(x/n)/sinh((n+1)/n) for x in range (n+2)]
```



Exemple pour $N = 15$

On représente ensuite les cellules par une liste de tableaux de la forme $[i, i+1]$ où i varie entre 0 et N.

```
Cells_1 = np.array([[i,i+1] for i in range (n+1)] )
```

Pour la matrice A_h, j'ai choisi de créer une matrice A de taille $(N+2) \times (N+2)$ pour traiter les cas aux bords, puis d'extraire la sous matrice A_h de taille $N \times N$ en enlevant les bords de A.

J'ai choisi d'utiliser la méthode de Simpson pour approcher l'intégrale $\int_{\hat{K}} \varphi_l'(\hat{x})\varphi_m'(\hat{x})d\hat{x}$.

On a alors

$$\int_{\hat{K}} \varphi_l'(\hat{x})\varphi_m'(\hat{x})d\hat{x} \simeq \frac{1}{6}\left(\varphi_l'(0)\varphi_m'(0) + 4\varphi_l'(1/2)\varphi_m'(1/2) + \varphi_l'(1)\varphi_m'(1)\right)$$

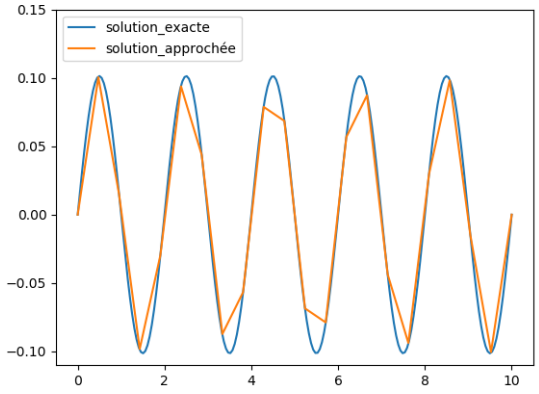
On peut donc calculer la matrice A en faisant une boucle sur les cellules, puis en faisant varier l et m dans l'ensemble $\{0, 1\}$ et en ajoutant $\dfrac{1}{x_{j+1} - x_j} \displaystyle\int_{\hat{K}} \varphi_l'(\hat{x})\varphi_m'(\hat{x})d\hat{x}$ à $A_{l,m}$.

De même, on calcule le vecteur colonne F de taille $N+2$, pour ensuite extraire F_h qui sont les N coordonnées de F en retirant les bords.

Enfin on résout le système $A_h U_h = F_h$ pour obtenir notre solution à l'aide la bibliothèque `numpy` et de la commande `Uh = np.linalg.solve(Ah, Fh)`.

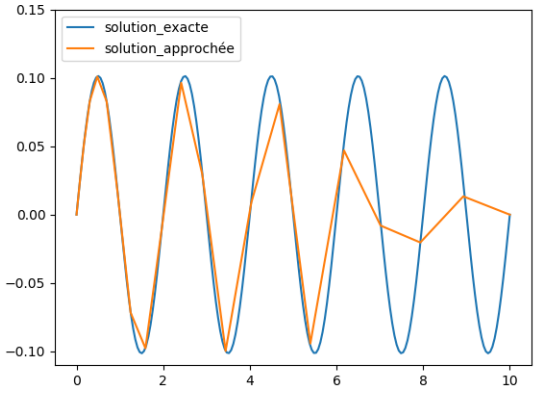
On présente maintenant les résultats pour les deux maillages présentés ci-dessus, pour $N \in \{20, 60, 200\}$.

Maillage uniforme

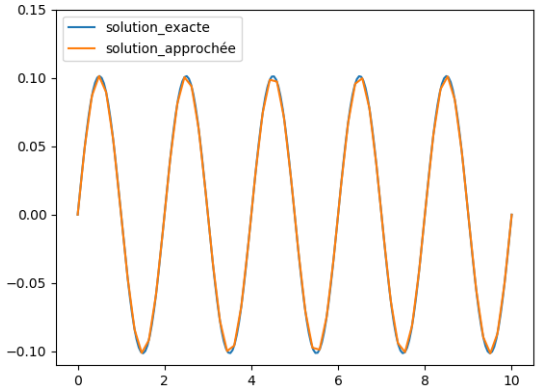


$N = 20$

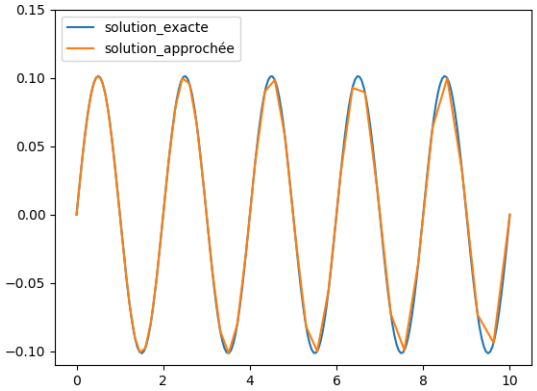
Maillage non uniforme



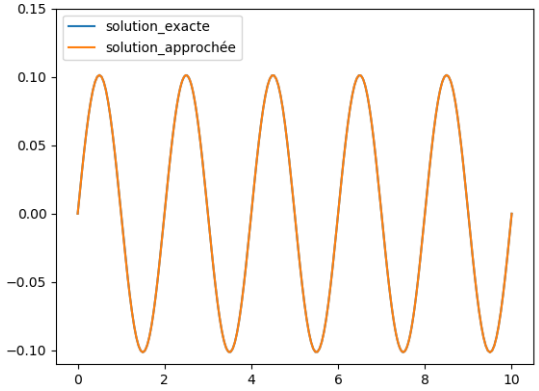
$N = 20$



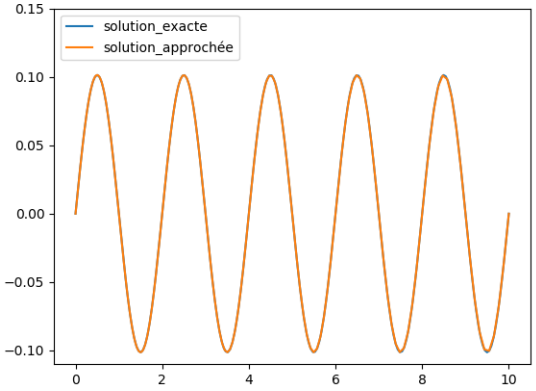
$N = 60$



$N = 60$



$N = 200$



$N = 200$

La méthode semble donc converger. Dans la partie suivante, on va quantifier cette convergence et calculer l'ordre de la convergence pour les normes L^2 et H^1.

3.3 Analyse numérique

On veut donner une estimation de l'erreur de notre approximation.

3.3.1 En théorie

Lemme 1 (de Céa)**.** Pour u la solution de (1), u_h solution de (3) et $\Pi_h(u)$ la projection de u sur l'espace $V_{h,1}$, on a

$$\|u - u_h\|_{H^1} \leqslant \frac{M}{\alpha}\|u - \Pi_h(u)\|_{H^1}$$

où M est la constante de continuité de a, α est la constante de coercivité de a avec a la forme bilinéaire de (2).

Démonstration. On a $u_h - \Pi_h(u) \in V_{h,1}$ par définition de u_h et $\Pi_h(u)$.

Donc on a

$$a(u, u_h - \Pi_h(u)) = b(u_h - \Pi_h(u)) \text{ et } a(u_h, u_h - \Pi_h(u)) = b(u_h - \Pi_h(u)),$$

puisque u et u_h sont solutions de (1) et (3).

En soustrayant, on a donc

$$a(u-u_h, u_h - \Pi_h(u)) = 0.$$

Par coercivité de a, on a

$$\begin{aligned}\alpha\|u-u_h\|_{H^1}^2 &\leqslant a(u-u_h, u-u_h) + \underbrace{a(u-u_h, u_h-\Pi_h(u))}_{=0} \\ &\leqslant a(u-u_h, u-\Pi_h(u)) \\ &\leqslant M\|u-u_h\|_{H^1}\|u-\Pi_h(u)\|_{H^1}\end{aligned}$$

Ainsi on a

$$\|u-u_h\|_{H^1} \leqslant \frac{M}{\alpha}\|u-\Pi_h(u)\|_{H^1}.$$

□

Théorème 1. Pour u solution $H^2(I)$ de (1) et $\Pi_h(u)$ la projection de u sur l'espace $V_{h,1}$, on a

$$\|u-\Pi_h(u)\|_{H^1(I)} \leqslant Ch\|u''\|_{L^2(I)}$$

où $h = \max_{[x_i,x_{i+1}]\in\tau_h} |x_{i+1}-x_i|$.

Démonstration.

Étape 1 : Montrons que $\|u' - \Pi_h(u)'\|_{L^2} \leqslant h\|u''\|_{L^2}$.

On va raisonner par densité, on suppose ici que u est $\mathcal{C}^\infty$.

$$\|u' - \Pi_h(u)'\|_{L^2}^2 = \sum_{i=0}^{N}\int_{x_i}^{x_{i+1}} (u'(x) - \Pi_h(u)'(x))^2 dx \tag{5}$$

Or, pour $x \in [x_i, x_{i+1}]$,

$$\Pi_h(u)'(x) = \frac{u(x_{i+1})-u(x_i)}{h_i} = u'(y_i)$$

avec $h_i = x_{i+1}-x_i$ et $y_i \in]x_i, x_{i+1}[$ défini par le théorème des accroissements finis.

$$\begin{aligned}\int_{x_i}^{x_{i+1}} (u'(x)-\Pi_h(u)'(x))^2 dx &= \int_{x_i}^{x_{i+1}} (u'(x)-u'(y_i))^2 dx \\ &= \int_{x_i}^{x_{i+1}} \left(\int_{y_i}^{x} u''(t)dt\right)^2 dx \\ &\leqslant \int_{x_i}^{x_{i+1}} \left(\int_{y_i}^{x} u''(t)^2 dt\right)\left(\int_{y_i}^{x} 1 dt\right) dx \text{ par Cauchy–Schwarz} \\ &\leqslant \int_{x_i}^{x_{i+1}} h_i \left(\int_{y_i}^{x} u''(t)^2 dt\right) dx \\ &\leqslant \int_{x_i}^{x_{i+1}} h_i \left(\int_{x_i}^{x_{i+1}} u''(t)^2 dt\right) dx \\ &\leqslant h_i \int_{x_i}^{x_{i+1}} u''(t)^2 \left(\int_{x_i}^{x_{i+1}} dx\right) dt \text{ par Fubini} \\ &\leqslant h_i^2 \int_{x_i}^{x_{i+1}} u''(t)^2 dt \end{aligned} \tag{6}$$

Ainsi, en injectant dans l'égalité (5), on a

$$\|u' - \Pi_h(u)'\|_{L^2}^2 \leqslant \left(\max_{i\in[\![0,N]\!]} h_i\right)^2 \|u''\|_{L^2}^2.$$

En passant à la racine, on a

$$\|u' - \Pi_h(u)'\|_{L^2} \leqslant h\|u''\|_{L^2}.$$

Étape 2 : Montrons que $\|u - \Pi_h(u)\|_{L^2} \leqslant h^2\|u''\|_{L^2}$.

$$\|u - \Pi_h(u)\|_{L^2}^2 = \sum_{i=0}^{N} \int_{x_i}^{x_{i+1}} |u(x) - \Pi_h(u)(x)|^2 dx$$

Or, pour $x \in [x_i, x_{i+1}]$,

$$\begin{aligned}
|u(x) - \Pi_h(u)(x)|^2 &= \left(\int_{x_i}^{x} (u'(t) - \Pi_h(u)'(t))dt\right)^2 \text{ car } u(x_i) = \Pi_h(u)(x_i) \\
&\leqslant h_i \int_{x_i}^{x} (u'(t) - \Pi_h(u)'(t))^2 dt \text{ par Cauchy–Schwarz} \\
&\leqslant h_i^3 \int_{x_i}^{x_{i+1}} u''(t)^2 dt \text{ par l'inégalité (6)}
\end{aligned}$$

On intègre sur $[x_i, x_{i+1}]$ pour obtenir

$$\int_{x_i}^{x_{i+1}} |u(x) - \Pi_h(u)(x)|^2 dx \leqslant h_i^4 \int_{x_i}^{x_{i+1}} u''(t)^2 dt.$$

Ainsi

$$\|u - \Pi_h(u)\|_{L^2}^2 \leqslant h^4\|u''\|_{L^2}^2.$$

En passant à la racine carrée, on obtient

$$\|u - \Pi_h(u)\|_{L^2} \leqslant h^2\|u''\|_{L^2}.$$

Étape 3 : Conclusion.

$$\begin{aligned}
\|u - \Pi_h(u)\|_{H^1} &= \|u - \Pi_h(u)\|_{L^2} + \|u' - \Pi_h(u)'\|_{L^2} \\
&\leqslant h^2\|u''\|_{L^2} + h\|u''\|_{L^2} \\
&\leqslant (1+h)h\|u''\|_{L^2} \\
&\leqslant Ch\|u''\|_{L^2} \text{ avec } C = |I| + 1
\end{aligned}$$

□

Corollaire 1. Dans notre cadre ($k = 1$), on a

$$\boxed{\|u - u_h\|_{L^2(I)} \leqslant Ch^2}$$

$$\boxed{\|u - u_h\|_{H^1(I)} \leqslant Ch}$$

où C dépend uniquement du domaine I et de la fonction f.

Démonstration. On a, par le lemme de Céa et par le fait que $-u'' = f$, les inégalités suivantes :

$$\begin{aligned}\|u - u_h\|_{L^2(I)} &\leqslant \frac{M}{\alpha}\|u - \Pi_h(u)\|_{L^2(I)} \\ &\leqslant C'h^2\|u''\|_{L^2} \text{ en utilisant la preuve du théorème 1} \\ &\leqslant C'h^2\|f\|_{L^2} \\ &\leqslant C''h^2 \text{ avec } C'' = C'\|f\|_{L^2}\end{aligned}$$

et

$$\begin{aligned}\|u - u_h\|_{H^1(I)} &\leqslant \frac{M}{\alpha}\|u - \Pi_h(u)\|_{H^1(I)} \\ &\leqslant C'h\|u''\|_{L^2} \text{ avec } C' = \frac{MC}{\alpha} \\ &\leqslant C'h\|f\|_{L^2} \\ &\leqslant C''h \text{ avec } C'' = C'\|f\|_{L^2}\end{aligned}$$

□

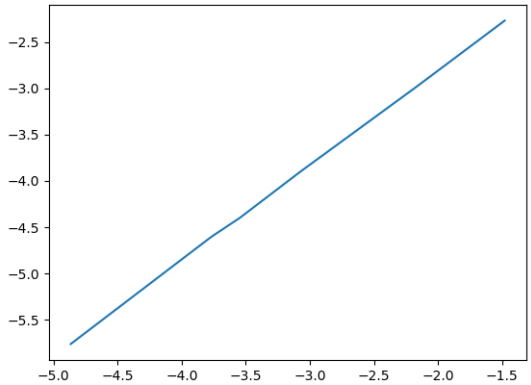
3.3.2 En pratique

On va maintenant calculer numériquement les erreurs entre la solution exacte et la solution approchée en normes L^2 et H^1. Pour cela, on calcule la norme L^2 de $u - u_h$ [1] en utilisant la fonction `quad` du module `scipy.integrate` de Python. On va donner la fonction u_h explicitement grâce à la fonction `uh_k1` (voir annexe 5.2). De plus, pour calculer la norme H^1, il faut calculer u_h' qui sera une fonction constante par morceaux qui vaut sur chaque $[x_i, x_{i+1}]$ du maillage le coefficient directeur de la pente entre $u_h(x_i)$ et $u_h(x_{i+1})$. On donne cette fonction explicitement grâce à `uhprim_k1` (voir annexe 5.2).

Ensuite pour calculer l'erreur, on va tracer la fonction $\log(\|u - u_h\|)$ en fonction de $\log h$. On va trouver une pente qui aura pour coefficient directeur l'exposant de h dans les inégalités que l'on a démontré dans la partie 3.3.1. On va tracer cette courbe pour $N \in \{100, 200, 500, 800, 1000, 3000\}$.

On va de nouveau faire une distinction entre les cas où le maillage sera uniforme ou non uniforme.

Les tracés sont de la forme :



Erreur en norme $H^1(I)$ pour le maillage non uniforme.

1. où u est la solution exacte et u_h la solution approchée calculée par la méthode des éléments finis

On résume les résultats en donnant le coefficient directeur moyen des pentes pour les différentes simulations.

Coefficient directeur	Maillage uniforme	Maillage non uniforme	Théorique
Norme L^2	1.9827103	2.0032377	2
Norme H^1	1.0016067	1.0153032	1

Ainsi, la simulation numérique coïncide bien avec les résultats théoriques.

4 Méthode de Lagrange pour des polynômes de degré au plus 2

4.1 Théorie

On va maintenant changer d'espace d'approximation, en élevant le degré des polynômes que l'on considère.

On se fixe un maillage τ_h de I.

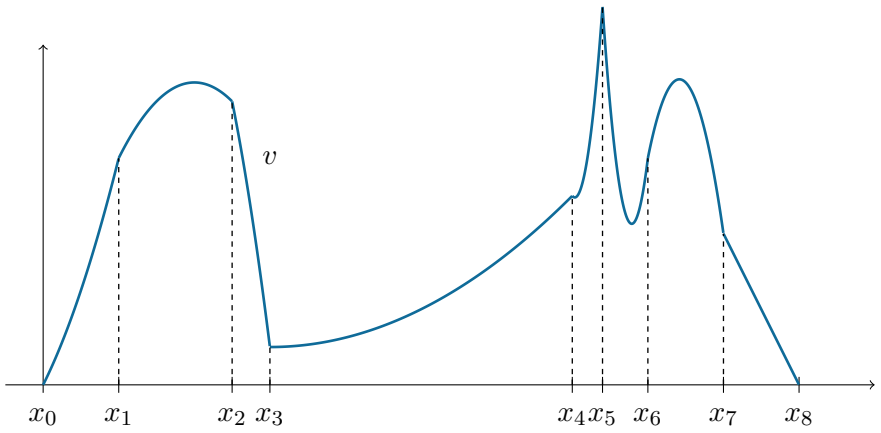
Définition 7. L'espace d'approximation $V_{h,2}$ que l'on va utiliser ici sera

$$V_{h,2} = \left\{ v \in \mathcal{C}^0(I), \quad v_{|[x_j,x_{j+1}]} \in \mathbb{P}_2[X], \quad j = 0, \dots, N \text{ avec } v(0) = v(10) = 0 \right\}$$

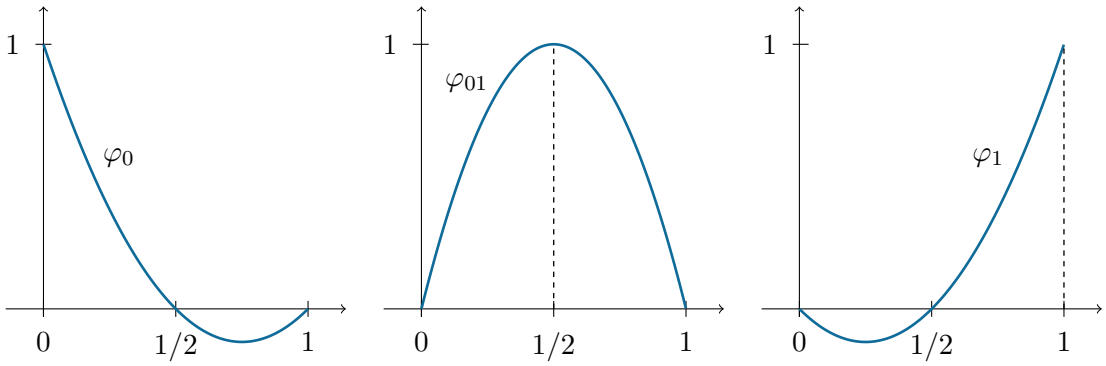
où $\mathbb{P}_2[X]$ désigne les polynômes réels de degré au plus 2.

Proposition 8. L'espace $V_{h,2}$ est de dimension $2N + 1$.

Exemple 9. La fonction v dessinée ci-dessous est dans $V_{h,2}$. (pour $N = 7$)



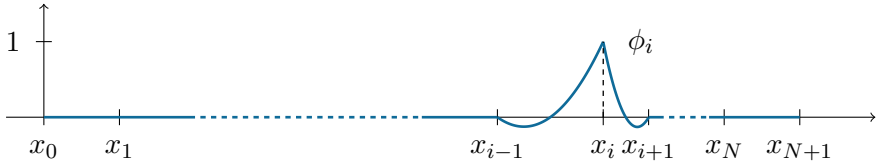
Il faut se donner une base de $V_{h,2}$. Pour chaque cellule $K = [x_j, x_{j+1}]$, on va rajouter un point au milieu $\widetilde{x_j}$, afin d'avoir trois points sur chaque cellule. Ainsi, comme il existe un unique polynôme de degré au plus deux qui passe par trois points distincts, on va pouvoir trouver 3 polynômes pour chaque cellule tels que chaque polynôme vaut 1 sur un des trois points et 0 sur les deux autres. On décrit les trois fonctions φ_0, φ_{01} et φ_1 pour la cellule de référence $\hat{K} = [0, 1]$.



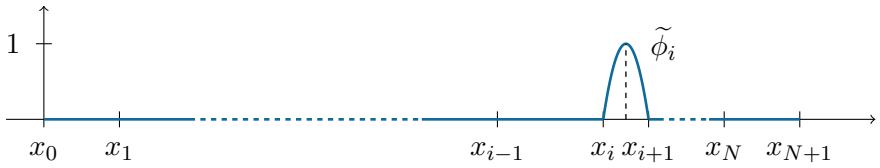
On peut ensuite utiliser les mêmes calculs que pour la méthode de Lagrange avec des polynômes de degré au plus 1, il faut juste faire varier m, l dans $\{0, 1, 2\}$ dans les intégrales que l'on considère.

Si on veut représenter la base de $V_{h,2}$ que l'on s'est donnée, on tracerait les $2N + 1$ fonctions suivantes (car $V_{h,2}$ est de dimension $2N + 1$).

On a les N fonctions $(\phi_i)_{i=1}^N$:



Puis on a les $N + 1$ fonctions $(\widetilde{\phi}_i)_{i=0}^N$:



4.2 Implémentation numérique

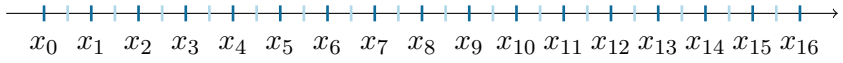
On veut toujours résoudre l'équation

$$\begin{cases} -u''(x) = \sin(\pi x) \quad \text{sur } [0, 10] \\ u(0) = u(10) = 0 \end{cases} \tag{7}$$

J'ai choisi d'implémenter le maillage par une liste qui contient les $2N + 3$ points rangés telle que les N points du maillage sont au début (rangés dans l'ordre croissant), puis on met les $N + 1$ points milieux (par ordre croissant) et enfin les deux bords.

Pour un maillage uniforme, on aura

```
Nodes_2_uni = Nodes_1_uni[1:-1]
        + [(Nodes_1_uni[i] + Nodes_1_uni[i+1])/2 for i in range (n+1)] + [0,10]
```



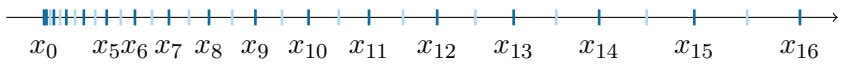
Exemple pour $N = 15$

```
Nodes = [1.66, 3.33, 5.0, 6.66, 8.33, 0.83, 2.5, 4.16, 5.83, 7.5, 9.16, 0, 10]
```

Exemple pour $N = 5$

On peut se donner un maillage non uniforme comme

```
Nodes_2 = Nodes_1[1:-1] + [(Nodes_1[i] + Nodes_1[i+1])/2 for i in range (n+1)]
        + [0,10]
```



Exemple pour $N = 15$

```
Nodes = [0.22, 0.90, 2.10, 3.92, 6.48, 0.11, 0.56, 1.50, 3.01, 5.20, 8.24, 0, 10]
```

Exemple pour $N = 5$

On représente ensuite les cellules par une liste de tableaux de la forme $[i, i+1, i+n+1]$ où i varie entre 0 et $N-2$. Ainsi on place le début de la cellule, la fin de la cellule puis le milieu de la cellule. Les deux cellules du bords sont donc de la forme `[31, 0, 15]` et `[14, 32, 30]` dans l'exemple où $N = 15$.

```
Cells_2 = [[2*n+1,0,n]] + [[i,i+1,i+n+1] for i in range(0,n-1)]+[[n-1,2*n+2,2*n]]
```

Je me suis donné un tableau `Bords` qui contient les deux bords :

```
Bords = [2*n+1, 2*n+2]
```

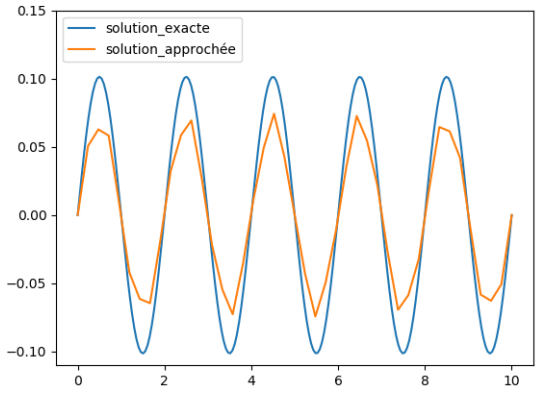
Pour la matrice A_h, j'ai choisi de créer une matrice A_h de taille $N \times N$ cette fois-ci et de gérer les bords dans la boucle de calcul des coefficients.

On peut donc calculer la matrice A_h en faisant une boucle sur les cellules, puis en faisant varier l et m dans l'ensemble $\{0, 1, 2\}$ et en ajoutant $\dfrac{1}{x_{j+1} - x_j} \int_{\hat{K}} \varphi_l'(\hat{x}) \varphi_m'(\hat{x}) d\hat{x}$ à $A_{l,m}$ si les indices correspondant à la cellule ne sont pas dans le tableau `Bords`.
De même, on calcule le vecteur colonne F_h de taille N en faisant attention aux bords.

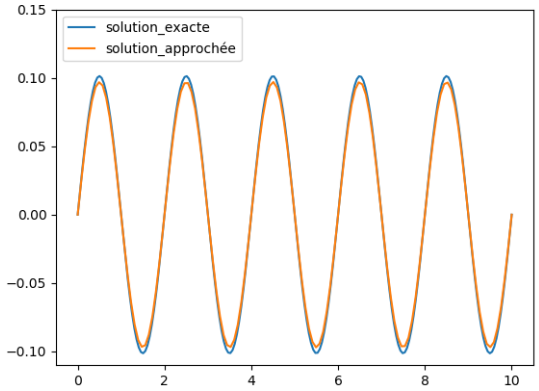
Enfin on résout le système $A_h U_h = F_h$ pour obtenir notre solution à l'aide la bibliothèque `numpy` et de la commande `Uh = np.linalg.solve(Ah, Fh)`.

On présente maintenant les résultats pour les deux maillages présentés ci-dessus, pour $N \in \{20, 60, 200\}$.

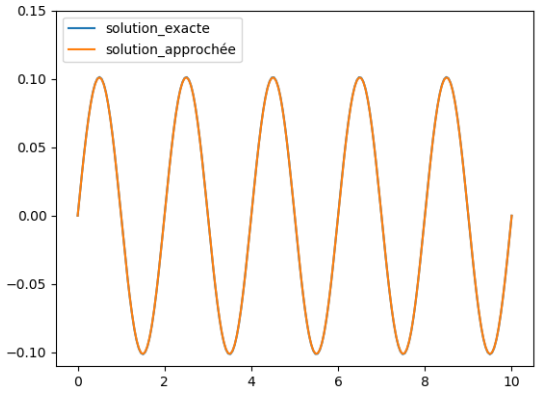
Maillage uniforme



$N = 20$

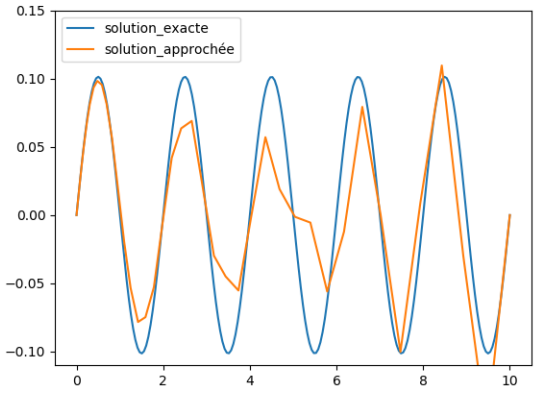


$N = 60$

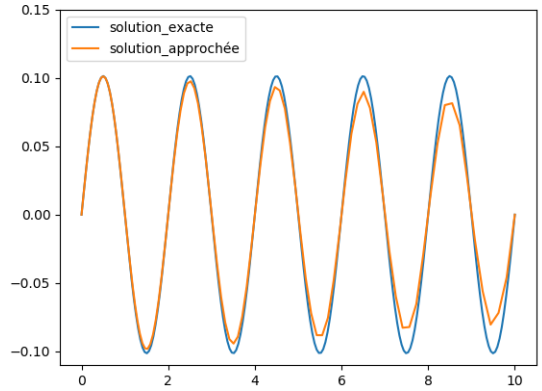


$N = 200$

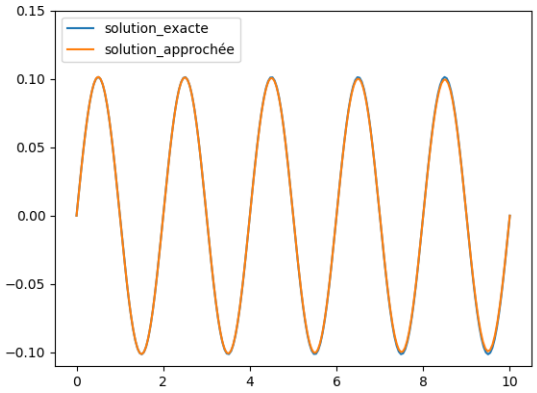
Maillage non uniforme



$N = 20$



$N = 60$



$N = 200$

Comme dans la partie précédente, la méthode semble converger. Dans la section suivante, on va quantifier cette convergence et calculer l'ordre de la convergence pour les normes L^2 et H^1.

4.3 Analyse numérique

On veut donner une estimation de l'erreur de notre approximation.

4.3.1 En théorie

Théorème 2. Pour u solution $H^3(I)$ de (1) et $\Pi_h(u)$ la projection de u sur l'espace $V_{h,2}$, on a

$$\|u - \Pi_h(u)\|_{H^1(I)} \leqslant Ch^2 \|u^{(3)}\|_{L^2(I)}$$

où $h = \max_{[x_i, x_{i+1}] \in \tau_h} |x_{i+1} - x_i|$.

Démonstration. Semblable à la preuve du théorème 1. Pour la preuve détaillée, voir l'annexe 5.5. □

Corollaire 2. Dans notre cadre ($k = 2$), on a

$$\boxed{\|u - u_h\|_{L^2(I)} \leqslant Ch^3}$$

$$\boxed{\|u - u_h\|_{H^1(I)} \leqslant Ch^2}$$

où C dépend uniquement du domaine I et de la fonction f.

Démonstration. Semblable à la preuve du corollaire 1. □

4.3.2 En pratique

On va maintenant calculer numériquement les erreurs entre la solution exacte et la solution approchée en normes L^2 et H^1. Pour cela, on calcule la norme L^2 de $u - u_h$ [2] en utilisant toujours la fonction `quad` du module `scipy.integrate` de Python. On va donner la fonction u_h explicitement grâce à la fonction `uh_k2` (voir annexe 5.4) en interpolant entre les 3 points $x_i, \widetilde{x_i}$ et x_{i+1}. De plus, pour calculer la norme H^1, il faut calculer u_h' qui sera une fonction affine par morceaux. On donne cette fonction explicitement grâce à `uhprim_k2` (voir annexe 5.4).

Ensuite pour calculer l'erreur, on va tracer la fonction $\log(\|u - u_h\|)$ en fonction de $\log h$. On va trouver une pente qui aura pour coefficient directeur l'exposant de h dans les inégalités que l'on a démontré dans la partie 3.3.1. On va tracer cette courbe pour $N \in \{100, 200, 500, 800, 1000, 3000\}$.

On va continuer de faire une distinction entre les cas où le maillage sera uniforme ou non uniforme.

On résume les résultats en donnant le coefficient directeur moyen des pentes pour les différentes simulations.

Coefficient directeur	Maillage uniforme	Maillage non uniforme	Théorique
Norme L^2	1.9922769	2.0006769	3
Norme H^1	2.0075953	2.0160268	2

2. où u est la solution exacte et u_h la solution approchée calculée par la méthode des éléments finis

On a donc une différence entre les résultats théoriques et les simulations pour la norme L^2. Cependant, pour la norme H^1, les simulations coïncident avec les résultats théoriques.

La différence de résultats pour la norme L^2 vient sans doute d'une erreur d'approximation d'intégrale que l'on effectue dans la méthode des éléments finis. En effet, dans la partie théorique, on calcule l'erreur pour la fonction u_h qui est donnée par la résolution du système $A_h U_h = F_h$. Or, dans la pratique, pour calculer A_h, on a utilisé une méthode de Simpson qui introduit une erreur de calcul numérique. Cette erreur a été négligée dans la partie théorique.

5 Annexe

5.1 Code pour le cas $k = 1$

```
from math import *
import numpy as np
from scipy.integrate import quad
import matplotlib.pyplot as plt


n=20
a=0
b=10

def f(x):
    return sin(pi*x)

def solution_exacte(x):
    return f(x)/(pi*pi)


Nodes_1 = [b*i/(n+1) + (1-i/(n+1))*a for i in range(n+2)]
#Nodes_1 = [x*b/(n+1)* sinh(x/n)/sinh((n+1)/n) for x in range (n+2)]

Cells_1 = np.array([[i,i+1] for i in range (n+1)] )


def RefBasicFct(x, k):
    if k == 1:
        return np.array([x, 1-x])
    if k == 2:
        return np.array([2*(x-1/2)*(x-1), 2*(x-1/2)*x, -4*x*(x-1)])

def DerRefBasisFct(x, k):
    if k == 1:
        return np.array([1,-1])
    if k == 2:
        return np.array([4*x - 3, 4*x - 1, -8*x + 4])
```

```
#Calcul de A_h
A = np.zeros((n+2,n+2))
for k in range (len(Cells_1)):
    for i in range(2):
        for j in range(2):
            A[Cells_1[k][i]][Cells_1[k][j]] += 1/(abs(Nodes_1[Cells_1[k][1]]-
                Nodes_1[Cells_1[k][0]]))*6*(DerRefBasisFct(0, 1)[i]*
                DerRefBasisFct(0, 1)[j]+4*DerRefBasisFct(1/2, 1)[i]*
                DerRefBasisFct(1/2, 1)[j]+DerRefBasisFct(1, 1)[i]*DerRefBasisFct(1, 1)[j])
A= np.array(A)
A = A[1:-1,1:-1]


#Calcul de F_h
F = np.zeros(n+2)
for k in range (len(Cells_1)):
    for i in range(2):
        F[Cells_1[k][i]] += (Nodes_1[Cells_1[k][1]]-Nodes_1[Cells_1[k][0]])*6*
             (RefBasicFct(0,1)[i]*f(Nodes_1[Cells_1[k][1]])+4*RefBasicFct(1/2,1)[i]*
             f(1/2*Nodes_1[Cells_1[k][0]]+(1/2)*Nodes_1[Cells_1[k][1]])+
             RefBasicFct(1,1)[i]*f(Nodes_1[Cells_1[k][0]]))
F = np.array(F)
F = F[1:-1]


#solution approchée
U = np.linalg.solve(A, F)

sol = np.zeros(n+2)
for i in range (1,n+1):
    sol[i] = U[i-1]

#tracer les courbes
plt.clf()
abscisse = np.linspace(a,b,200)
plt.plot(abscisse,[solution_exacte(x) for x in abscisse], label="solution_exacte")
plt.plot(Nodes_1,sol, label="solution_approchée")
plt.legend(loc='upper left')
plt.ylim(-0.11, 0.15)
plt.show()
```

5.2 Code pour l'erreur dans le cas $k = 1$

```
def moy_coef(a,b):
    S = 0
    for i in range (len(a)-1):
```

```
        S+= ((b[i+1]-b[i])/(a[i+1]-a[i]))
    return S/(len(a)-1)

def uh_k1(L, Nodes, x):
    i = 0
    while Nodes[i] < x:
        i += 1
    a = Nodes[i-1]
    b = Nodes[i]
    fa = L[i-1]
    fb = L[i]
    return fb/(b-a)*(x-a) + fa/(a-b)*(x-b)

#resol_k1 donne le vecteur U_h
def erreur_k1_L2(n):
    Nodes_1 = [b*i/(n+1) + (1-i/(n+1))*a for i in range(n+2)]
    # Nodes_1 = [x*b/(n+1)* sinh(x/n)/sinh((n+1)/n) for x in range (n+2)]
    sa = resol_k1(n)
    def g(x):
        return uh_k1(sa,Nodes_1,x)
    return sqrt(quad(lambda x : (g(x)-f(x)/(pi*pi))**2,a,b)[0])


def uhprim_k1(L, Nodes, x):
    i = 0
    while Nodes[i] < x:
        i += 1
    a = Nodes[i-1]
    b = Nodes[i]
    fa = L[i-1]
    fb = L[i]
    return (fb - fa) / (b - a)

def erreur_k1_H1(n):
    Nodes_1 = [b*i/(n+1) + (1-i/(n+1))*a for i in range(n + 2)]
    #Nodes_1 = [x*b/(n+1)* sinh(x/n)/sinh((n+1)/n) for x in range (n + 2)]
    sa = resol_k1(n)
    def g(x):
        return uh_k1(sa, Nodes_1, x)
    def gprim(x):
        return uhprim_k1(sa, Nodes_1, x)
    return sqrt(quad(lambda x : (g(x)-f(x)/(pi*pi))**2, a, b)[0])
             +sqrt(quad(lambda x : (gprim(x)-cos(x*pi)/(pi))**2, a, b)[0])

def trace_erreur_k1(norm):
    xlinspace = np.array([100,200,500,800, 1000, 3000])
    def maxliste(L):
        max = L[1] - L[0]
```

```
        for i in range (len(L) - 1):
            if (L[i+1] - L[i]) > max :
                max = L[i+1] - L[i]
        return max
    #H = [maxliste([x*b/(n+1)* sinh(x/n)/sinh((n+1)/n) for x in range (n+2)])
                                                                    for n in xlinspace]
    H = [maxliste([b*i/(n+1) + (1-i/(n+1))*a for i in range(n+2)]) for n in xlinspace]
    if norm == "L2":
        Erreur = [erreur_k1_L2(n) for n in xlinspace]
        plt.clf()
        plt.plot([log(h) for h in H], [log(x) for x in Erreur])
        plt.show()
    if norm == "H1":
        Erreur = [erreur_k1_H1(n) for n in xlinspace]
        plt.clf()
        plt.plot([log(h) for h in H], [log(x) for x in Erreur])
        plt.show()
    return moy_coef([log(h) for h in H], [log(x) for x in Erreur])
```

5.3 Code pour le cas $k = 2$

```
from math import *
import numpy as np
from scipy.integrate import quad
import matplotlib.pyplot as plt

n=20
a=0
b=10



def f(x):
    return sin(pi*x)

def solution_exacte(x):
    return f(x)/(pi*pi)


Nodes_1 = [b*i/(n+1) + (1-i/(n+1))*a for i in range(n+2)]
#Nodes_1 = [x*b/(n+1)* sinh(x/n)/sinh((n+1)/n) for x in range (n+2)]

Nodes_2 = Nodes_1[1:-1] + [(Nodes_1[i] + Nodes_1[i+1])/2 for i in range (n+1)] + [a,b]

Cells_2 = [[2*n+1,0,n]]+[[i,i+1,i+n+1] for i in range(0,n-1)]+[[n-1,2*n+2,2*n]]

Bords = [2*n+1, 2*n+2]
```

```
def RefBasicFct(x, k):
    if k == 1:
        return np.array([x, 1-x])
    if k == 2:
        return np.array([2*(x-1/2)*(x-1), 2*(x-1/2)*x, -4*x*(x-1)])

def DerRefBasisFct(x, k):
    if k == 1:
        return np.array([1,-1])
    if k == 2:
        return np.array([4*x - 3, 4*x - 1, -8*x + 4])



#Calcul de A_h
A_2 = np.zeros((2*n+1,2*n+1))
for k in range (len(Cells_2)):
    for i in range(3):
        for j in range(3):
            I = Cells_2[k][i]
            J = Cells_2[k][j]
            if I not in Bords and J not in Bords:
                A_2[I][J] += 1/(abs(Nodes_2[Cells_2[k][1]]-Nodes_2[Cells_2[k][0]]))*6*
                     (DerRefBasisFct(0, 2)[i]*DerRefBasisFct(0, 2)[j]+4*
                     DerRefBasisFct(1/2, 2)[i]*DerRefBasisFct(1/2, 2)[j]+
                     DerRefBasisFct(1, 2)[i]*DerRefBasisFct(1, 2)[j])

A_2 = np.array(A_2)


#Calcul de F_h
F_2 = np.zeros(2*n+1)
for k in range (len(Cells_2)):
    for i in range(3):
        I = Cells_2[k][i]
        if I not in Bords:
            F_2[I] += (Nodes_2[Cells_2[k][1]]-Nodes_2[Cells_2[k][0]])*6*
                 (RefBasicFct(0,2)[i]*f(Nodes_2[Cells_2[k][1]])+
                 4*RefBasicFct(1/2,2)[i]*
                 f(1/2*Nodes_2[Cells_2[k][0]]+1/2*Nodes_2[Cells_2[k][1]])
                 +RefBasicFct(1,2)[i]* f(Nodes_2[Cells_2[k][0]]))

F_2 = np.array(F_2)
```

```
#Calcul de U_h
U_2 = np.linalg.solve(A_2, F_2)


def reordonne(a,n):
    V = np.zeros(len(a))
    for i in range(n):
        V[2*i+1] = a[i]
    for i in range(n+1):
        V[2*i] = a[i+n]
    return V

#réordonne et ajoute des zeros sur les bords
segment = np.concatenate((np.concatenate((np.array([a]),
      reordonne(Nodes_2[:-2],n))),np.array([b])))
solution = np.concatenate((np.concatenate((np.array([0]),
      reordonne(U_2,n))),np.array([0])))

#tracer les courbes
plt.clf()
abscisse = np.linspace(a,b,200)
plt.plot(abscisse,[solution_exacte(x) for x in abscisse],label="solution_exacte")
plt.plot(segment, solution, label="solution_approchée")
plt.legend(loc='upper left')
plt.ylim(-0.11, 0.15)
plt.show()
```

5.4 Code pour l'erreur dans le cas $k = 2$

```
def uh_k2(L, Nodes, x):
    i = 0
    while i < len(Nodes) and Nodes[i] < x:
        i += 1
    if i\%2==0:
        a = Nodes[i-2]
        b = Nodes[i-1]
        c = Nodes[i]
        fa = L[i-2]
        fb = L[i-1]
        fc = L[i]
    else:
        a = Nodes[i-1]
        b = Nodes[i]
        c = Nodes[i+1]
        fa = L[i-1]
        fb = L[i]
        fc = L[i+1]
```

```
    return fc/((c-a)*(c-b))*(x-a)*(x-b) + fa/((a-b)*(a-c))*(x-b)*(x-c)
               + fb/((b-a)*(b-c))*(x-a)*(x-c)

def erreur_k2_L2(n):
    Nodes_1 = [b*i/(n+1) + (1-i/(n+1))*a for i in range(n+2)]
    Nodes_2 = Nodes_1[1:-1] +
         [(Nodes_1[i] + Nodes_1[i+1])/2 for i in range (n+1)] + [a,b]
    sa = resol_k2(n)
    solution = np.concatenate((np.concatenate(
         (np.array([0]),sa)),np.array([0])))
    segment = np.concatenate((np.concatenate(
         (np.array([a]),reordonne(Nodes_2[:-2],n))),np.array([b])))
    def g(x):
        return uh_k2(solution,segment,x)
    return sqrt(quad(lambda x : (g(x) - f(x)/(pi*pi))**2,a,b)[0])

def uhprim_k2(L, Nodes, x):
    i = 0
    while i<len(Nodes) and Nodes[i] < x:
        i += 1
    if i\%2==0:
        a = Nodes[i-2]
        b = Nodes[i-1]
        c = Nodes[i]
        fa = L[i-2]
        fb = L[i-1]
        fc = L[i]
    else:
        a = Nodes[i-1]
        b = Nodes[i]
        c = Nodes[i+1]
        fa = L[i-1]
        fb = L[i]
        fc = L[i+1]
    return fc/((c-a)*(c-b))*(2*x-a-b) + fa/((a-b)*(a-c))*(2*x-b-c)
              + fb/((b-a)*(b-c))*(2*x-a-c)

def erreur_k2_H1(n):
    Nodes_1 = [b*i/(n+1) + (1-i/(n+1))*a for i in range(n+2)]
    Nodes_2 = Nodes_1[1:-1] +
         [(Nodes_1[i] + Nodes_1[i+1])/2 for i in range (n+1)] + [a,b]
    sa = resol_k2(n)
    solution = np.concatenate((np.concatenate(
         (np.array([0]),sa)),np.array([0])))
    segment = np.concatenate((np.concatenate(
         (np.array([a]),reordonne(Nodes_2[:-2],n))),np.array([b])))
    def g(x):
        return uh_k2(solution,segment,x)
```

```
    def gprim(x):
        return uhprim_k2(solution,segment,x)
    return sqrt(quad(lambda x : (g(x)-f(x)/(pi*pi))**2,a,b)[0])
              + sqrt(quad(lambda x : (gprim(x)-cos(x*pi)/(pi))**2,a,b)[0])

def trace_erreur_k2(norm):
    xlinspace = np.array([100,200,500,800, 1000, 3000])
    if norm == "infini":
        Erreur = [erreur_k2_infini(n) for n in xlinspace]
        plt.clf()
        plt.plot([log(1/n) for n in xlinspace],[log(x) for x in Erreur])
        plt.show()

    if norm == "L2":
        Erreur = [erreur_k2_L2(n) for n in xlinspace]
        plt.clf()
        plt.plot([log(1/n) for n in xlinspace],[log(x) for x in Erreur])
        plt.show()

    if norm == "H1":
        Erreur = [erreur_k2_H1(n) for n in xlinspace]
        plt.clf()
        plt.plot([log(1/n) for n in xlinspace],[log(x) for x in Erreur])
        plt.show()
    return moy_coef([log(1/n)for n in xlinspace],[log(x) for x in Erreur])
```

5.5 Preuve du théorème 2

On veut montrer que

$$\|u - \Pi_h(u)\|_{H^1} \leqslant Ch^2 \|u^{(3)}\|_{L^2}.$$

Étape 1 : Montrons que $\|u'' - \Pi_h(u)''\|_{L^2} \leqslant h\|u^{(3)}\|_{L^2}$.

On va raisonner par densité, on suppose ici que u est $\mathcal{C}^\infty$.

$$\|u'' - \Pi_h(u)''\|_{L^2}^2 = \sum_{i=0}^{N} \int_{x_i}^{x_{i+1}} (u''(x) - \Pi_h(u)''(x))^2 dx \tag{8}$$

Or, pour $x \in [x_i, x_{i+1}]$,

$$\Pi_h(u)(x) = \frac{2u(x_i)}{h_i^2}(x - x_{i+1})(x - \widetilde{x_i}) - \frac{4u(\widetilde{x_i})}{h_i^2}(x - x_i)(x - x_{i+1}) + \frac{2u(x_{i+1})}{h_i^2}(x - x_i)(x - \widetilde{x_i})$$

Ainsi, on a

$$\Pi_h(u)''(x) = \frac{4}{h_i^2}\left(u(x_i) - 2u(\widetilde{x_i}) + u(x_{i+1})\right) = u''(y_i)$$

où $y_i \in]x_i, x_{i+1}[$[3].

3. considérer $\dfrac{u(t+h) - 2u(t) + u(t-h)}{h^2}$ et appliquer le théorème de Rolle à une fonction bien choisie (similaire à la preuve du théorème des accroissements finis)

$$\begin{aligned}
\int_{x_i}^{x_{i+1}} (u''(x) - \Pi_h(u)''(x))^2 dx &= \int_{x_i}^{x_{i+1}} (u''(x) - u''(y_i))^2 dx \\
&= \int_{x_i}^{x_{i+1}} \left(\int_{y_i}^{x} u^{(3)}(t) dt \right)^2 dx \\
&\leqslant \int_{x_i}^{x_{i+1}} \left(\int_{y_i}^{x} u^{(3)}(t)^2 dt \right) \left(\int_{y_i}^{x} 1 dt \right) dx \text{ par Cauchy–Schwarz} \\
&\leqslant \int_{x_i}^{x_{i+1}} h_i \left(\int_{y_i}^{x} u^{(3)}(t)^2 dt \right) dx \\
&\leqslant \int_{x_i}^{x_{i+1}} h_i \left(\int_{x_i}^{x_{i+1}} u^{(3)}(t)^2 dt \right) dx \\
&\leqslant h_i \int_{x_i}^{x_{i+1}} u^{(3)}(t)^2 \left(\int_{x_i}^{x_{i+1}} dx \right) dt \text{ par Fubini} \\
&\leqslant h_i^2 \int_{x_i}^{x_{i+1}} u^{(3)}(t)^2 dt \qquad (9)
\end{aligned}$$

Ainsi, en injectant dans l'égalité (8), on a

$$\|u'' - \Pi_h(u)''\|_{L^2}^2 \leqslant h^2 \|u^{(3)}\|_{L^2}^2.$$

En passant à la racine, on a

$$\|u'' - \Pi_h(u)''\|_{L^2} \leqslant h \|u^{(3)}\|_{L^2}.$$

Étape 2 : Montrons que $\|u' - \Pi_h(u)'\|_{L^2} \leqslant h^2 \|u^{(3)}\|_{L^2}$.

$$\|u' - \Pi_h(u)'\|_{L^2}^2 = \sum_{i=0}^{N} \int_{x_i}^{x_{i+1}} |u'(x) - \Pi_h(u)'(x)|^2 dx$$

Or, pour $x \in [x_i, x_{i+1}]$,

$$\begin{aligned}
|u'(x) - \Pi_h(u)'(x)|^2 &= \left(\int_{x_i}^{x} (u''(t) - \Pi_h(u)''(t)) dt \right)^2 \\
&\leqslant h_i \int_{x_i}^{x} (u''(t) - \Pi_h(u)''(t))^2 dt \text{ par Cauchy–Schwarz} \\
&\leqslant h_i^3 \int_{x_i}^{x_{i+1}} u^{(3)}(t)^2 dt \text{ par l'inégalité (9)} \qquad (10)
\end{aligned}$$

On intègre sur $[x_i, x_{i+1}]$ pour obtenir

$$\int_{x_i}^{x_{i+1}} |u'(x) - \Pi_h(u)'(x)|^2 dx \leqslant h_i^4 \int_{x_i}^{x_{i+1}} u^{(3)}(t)^2 dt.$$

Ainsi

$$\|u' - \Pi_h(u)'\|_{L^2}^2 \leqslant h^4 \|u^{(3)}\|_{L^2}^2.$$

En passant à la racine carrée, on obtient

$$\|u' - \Pi_h(u)'\|_{L^2} \leqslant h^2 \|u^{(3)}\|_{L^2}.$$

Étape 3 : Montrons que $\|u - \Pi_h(u)\|_{L^2} \leqslant h^3 \|u^{(3)}\|_{L^2}$.

$$\|u - \Pi_h(u)\|_{L^2}^2 = \sum_{i=0}^{N} \int_{x_i}^{x_{i+1}} |u(x) - \Pi_h(u)(x)|^2 dx$$

Or, pour $x \in [x_i, x_{i+1}]$,

$$\begin{aligned}
|u(x) - \Pi_h(u)(x)|^2 &= \left(\int_{x_i}^{x} (u'(t) - \Pi_h(u)'(t)) dt \right)^2 \\
&\leqslant h_i \int_{x_i}^{x} (u'(t) - \Pi_h(u)'(t))^2 dt \text{ par Cauchy–Schwarz} \\
&\leqslant h_i \int_{x_i}^{x_{i+1}} h_i^3 \int_{x_i}^{x_{i+1}} u^{(3)}(s)^2 ds dt \text{ par l'inégalité (10)} \\
&\leqslant h_i^5 \int_{x_i}^{x_{i+1}} u^{(3)}(s)^2 ds
\end{aligned}$$

On intègre sur $[x_i, x_{i+1}]$ pour obtenir

$$\int_{x_i}^{x_{i+1}} |u(x) - \Pi_h(u)(x)|^2 dx \leqslant h_i^6 \int_{x_i}^{x_{i+1}} u^{(3)}(t)^2 dt.$$

Ainsi

$$\|u - \Pi_h(u)\|_{L^2}^2 \leqslant h^6 \|u^{(3)}\|_{L^2}^2.$$

En passant à la racine carrée, on obtient

$$\|u - \Pi_h(u)\|_{L^2} \leqslant h^3 \|u^{(3)}\|_{L^2}.$$

Étape 3 : Conclusion.

$$\begin{aligned}
\|u - \Pi_h(u)\|_{H^1} &= \|u - \Pi_h(u)\|_{L^2} + \|u' - \Pi_h(u)'\|_{L^2} \\
&\leqslant h^3 \|u^{(3)}\|_{L^2} + h^2 \|u^{(3)}\|_{L^2} \\
&\leqslant (1+h) h^2 \|u^{(3)}\|_{L^2} \\
&\leqslant C h^2 \|u^{(3)}\|_{L^2} \text{ avec } C = |I| + 1
\end{aligned}$$

5.6 Résultats plus généraux

Lemme 2 (Céa)**.** Pour u la solution de (1) dans un Hilbert V, u_h solution de (3), on a

$$\|u - u_h\|_V \leqslant \frac{M}{\alpha} \inf_{v_h \in V_h} \|u - v_h\|_V$$

où M est la constante de continuité de a, α est la constante de coercivité de a avec a la forme bilinéaire de (2).

Théorème 3. Pour u solution $H^{k+1}(I)$ de (1) et $\Pi_h(u)$ la projection de u sur l'espace V_h des polynômes de degré au plus k par morceaux sur un maillage τ_h, on a

$$\|u - \Pi_h(u)\|_{H^1(I)} \leqslant C h^k \|u^{(k+1)}\|_{L^2(I)}$$

où $h = \max_{[x_i, x_{i+1}] \in \tau_h} |x_{i+1} - x_i|$.