Notes de cours de l'ISIMA, deuxième année, http://www.isima.fr/~leborgne

Programmation des éléments finis P1 en 1D

Gilles Leborgne

12 janvier 2021

Table des matières

1 Le problème

1.1 Le problème initial

1.1.1 L'équation initiale

On se place sur un intervalle $[a,b] \subset \mathbb{R}$ avec $-\infty < a < b < \infty$.
Il s'agit de résoudre numériquement l'équation aux limites de Sturm–Liouville :

$$\left\{ \begin{array}{l} -(p(x)u'(x))' + q(x)u(x) = f(x), \quad \text{pour } x \in [a,b], \\ \text{C.L.}, \end{array} \right. \tag{1.1}$$

où p et q sont des fonctions positives et bornées.
Les conditions aux limites en a et b sont l'une des conditions suivantes :
— $u(x_0)$ et $u(b)$ sont connues (C.L. de Dirichlet),
— $u'(a)$ et $u'(b)$ sont connues (C.L. de Neumann),
— $u(a)$ et $u'(b)$ sont connues (C.L. mixtes Dirichlet–Neumann),
— Neumann généralisé : $p(x_0)u'(x_0) + \lambda_{x_0}u(x_0)$ connue (C.L. mixte), pour x_0=a et/ou b, et pour $\lambda_{x_0} \in \mathbb{R}$ donné.

1.1.2 La forme variationnelle

Formellement, on multiplie $(1.1)_1$ par une fonction v et on intègre sur $[a,b]$:

$$\left\{ \begin{array}{l} \displaystyle\int_a^b p(x)u'(x)v'(x)\,dx - [p(x)u'(x)v(x)]_a^b + \int_a^b q(x)u(x)v(x)\,dx = \int_a^b f(x)v(x)\,dx, \\ \text{C.L.}. \end{array} \right. \tag{1.2}$$

Prise en compte des conditions aux limites $(1.1)_2$: 2 cas.

1. Si u est connu en a (resp. en b), alors il n'y a pas à le calculer : il suffit de prendre dans (1.2) les v tels que $v(a) = 0$ (resp. $v(b) = 0$). C'est le cas des conditions de Dirichlet.
2. Sinon, u n'est pas connu en a (resp. en b), et il faut le calculer : on ne peut pas se contenter de considérer les v tels que $v(a) = 0$ (resp. $v(b) = 0$). C'est le cas des C.L. de Neumann et C.L. mixtes.

Exemple. On suppose $u(a) = g_a$ connu et $p(b)u'(b) + \lambda_b u(b) = g_b$ connu. Le terme de bord s'écrit alors, sachant qu'on ne prend que les v tels que $v(a) = 0$:

$$[p(x)u'(x)v(x)]_a^b = p(b)u'(b)v(b) - 0 = g_b v(b) - \lambda_b u(b)v(b).$$

D'où (1.2) s'écrit : trouver la fonction u telle que pour toute fonction v vérifiant $v(a) = 0$:

$$\int_a^b p(x)u'(x)v'(x)\,dx + \lambda_b u(b)v(b) + \int_a^b q(x)u(x)v(x)\,dx = \int_a^b f(x)v(x)\,dx + g_b v(b).$$

Ayant supposé p et $q \in L^\infty([a,b])$, l'équation ci-dessus n'est plus formelle dès que u et v sont pris dans $H^1([a,b])$, i.e. dès que u, v, u' et v' sont dans $L^2([a,b])$, i.e. de carré intégrable. Le problème (1.2) s'écrit donc rigoureusement :

$$\left\{ \begin{array}{l} \text{Trouver } u \in H^1([a,b]) \text{ où } u(a) = u_a \text{ t.q. } \forall v \in H^1([a,b]) \text{ où } v(a) = 0 : \\ \underbrace{\displaystyle\int_a^b p(x)u'(x)v'(x)\,dx + \int_a^b q(x)u(x)v(x)\,dx + \lambda_b u(b)v(b)}_{a(u,v) \text{ bilinaire}} = \underbrace{\displaystyle\int_a^b f(x)v(x)\,dx + g_b v(b)}_{\ell(v) \text{ linaire}}. \end{array} \right. \tag{1.3}$$

Exercice 1.1 Réciproquement, montrer qu'une solution de (1.3) satisfait à (1.1) au sens des distributions. ∎

1.2 Le problème discret associé

On souhaite trouver une solution approchée (numérique) u_h de u. On va chercher ici une solution $u_h \in P_1$, i.e. qui est continue sur $[a,b]$ et affine par morceaux.

1.2.1 Les fonctions de base pour le problème discret

L'espace P_1 des éléments finis : on commence par partager $[a,b]$ en n intervalles : $[a,b] = \bigcup_{i=1}^{n} I_k$ où $I_k = [x_{k-1}, x_k]$ avec $a = x_0 < x_1 ... < x_n = b$. Et $P_1 = \{f \in C^0([a,b];\mathbb{R}) : f_{|I_i} \text{ affine}\}$ = l'ensemble des fonctions continues sur $[a,b]$ qui sont affines sur chaque I_k.

Base usuelle de P_1 : les $n+1$ fonctions chapeaux $\varphi_i \in P_1$ définies par :

$$\forall i,j = 0,...,n,\ \varphi_i(x_j) = \delta_{ij}.$$

Donc :

$$\begin{cases} \varphi_0(x) = \dfrac{x_1 - x}{x_1 - x_0} \text{ dans } [x_0, x_1], \\ \varphi_0(x) = 0 \text{ dans } [x_1, x_n], \end{cases}$$

$$\forall i = 1,...,n-1, \quad \begin{cases} \varphi_i(x) = 0 \text{ dans } [x_0, x_{i-1}] \bigcup [x_{i+1}, x_n], \\ \varphi_i(x) = \dfrac{x - x_{i-1}}{x_i - x_{i-1}} \text{ dans } [x_{i-1}, x_i], \\ \varphi_i(x) = \dfrac{x_{i+1} - x}{x_{i+1} - x_i} \text{ dans } [x_i, x_{i+1}], \end{cases}$$

$$\begin{cases} \varphi_n(x) = 0 \text{ dans } [x_0, x_{n-1}], \\ \varphi_n(x) = \dfrac{x - x_{n-1}}{x_n - x_{n-1}} \text{ dans } [x_{n-1}, x_n]. \end{cases}$$

Dessin. (Exercice facile : c'est bien une base.) Si $g \in P_1$, alors notons c_i ses composantes sur la base $(\varphi_i)_{i=0,...,n}$, i.e. écrivons

$$g(x) = \sum_{j=0}^{n} c_j \varphi_j(x), \quad \text{où donc} \quad c_i = g(x_i) \text{ pour } i = 0,...,n.$$

Et connaître une fonction $g \in P_1$, c'est connaître ses composantes c_j sur la base $(\varphi_i)_{i=0,...,n}$.

1.2.2 La non compatibilité du problème initial

On cherche une solution $u_h \in P_1$ de (1.1). Cela semble absurde. En effet, u_h n'est pas dérivable aux points x_i, et il est encore plus ambitieux de considérer sa dérivée seconde : il y a apparition de masse de Dirac aux points x_i et (1.1) n'est plus une équation fonctionnelle.

1.2.3 Formulation variationnelle pour le problème approché

Ce n'est donc pas une solution P_1 de (1.1) qu'on cherche, mais une solution P_1 de (1.3). C'est sensé car les fonctions $f \in P_1$ sont bien dans l'espace $H^1([a,b])$. D'où le problème approché (avec problème de Dirichlet en a) :

$$\begin{cases} \text{Trouver } u_h \in P_1 \text{ où } u_h(a) = u_a \text{ t.q. } \forall v_h \in P_1 \text{ vérifiant } v_h(a) = 0 : \\ a(u_h, v_h) = \ell(v_h). \end{cases} \tag{1.4}$$

1.2.4 Calcul pratique de la solution approchée

Soit (avec la condition de Dirichlet imposée en a)

$$P_{1a} = \{v_h \in P_1 : v_h(a) = 0\}. \tag{1.5}$$

Les $(\varphi_i)_{i=1,...,n}$ forment une base de P_{1a}, et le problème (1.4) s'écrit de manière équivalente : résoudre le problème de n équations

$$\begin{cases} \text{Trouver } u_h \in P_1 \text{ où } u_h(a) = u_a \text{ t.q. pour tout } i = 1,...,n : \\ \displaystyle\int_a^b p(x) u_h'(x) \varphi_i'(x)\,dx + \int_a^b q(x) u_h(x) \varphi_i(x)\,dx + \lambda_b u_h(b) \varphi_i(b) = \int_a^b f(x) \varphi_i(x)\,dx + g_b \varphi_i(b). \end{cases} \tag{1.6}$$

Et connaître u_h, c'est connaître ses composantes c_j sur la base φ_i :

$$u_h(x) = \sum_{j=0}^{n} c_j \varphi_j(x), \quad \text{avec } c_0 = u_a \text{ connu : Dirichlet en } a.$$

Et (1.6) s'écrit donc comme le problème de n inconnues et n équations :

$$\left\{ \begin{array}{l} \text{Trouver les } c_j,\ j = 1, ..., n \text{ t.q. pour tout } i = 1, ..., n : \\ \displaystyle\sum_{j=0}^{n} (\int_a^b p(x)\varphi_j'(x)\varphi_i'(x)\,dx)\, c_j + \sum_{j=0}^{n} (\int_a^b q(x)\varphi_j(x)\varphi_i(x)\,dx) c_j + \lambda_b c_n \delta_{in} \\ \qquad\qquad = \displaystyle\int_a^b f(x)\varphi_i(x)\,dx + g_b \varphi_i(b). \end{array} \right. \tag{1.7}$$

On pose (générique pour toutes les conditions aux limites) :

$$\left\{ \begin{array}{l} R = [R_{ij}]_{\substack{0\le i\le n \\ 0\le j\le n}} = [\displaystyle\int_a^b p(x)\varphi_j'(x)\varphi_i'(x)\,dx]_{\substack{0\le i\le n \\ 0\le j\le n}}, \\ M = [M_{ij}]_{\substack{0\le i\le n \\ 0\le j\le n}} = [\displaystyle\int_a^b q(x)\varphi_j(x)\varphi_i(x)\,dx + \lambda_b c_n \delta_{in}]_{\substack{0\le i\le n \\ 0\le j\le n}}, \end{array} \right. \tag{1.8}$$

et :

$$\vec{c} = \begin{pmatrix} c_0 \\ \vdots \\ c_n \end{pmatrix}, \qquad \vec{f} = \begin{pmatrix} \int_a^b f(x)\varphi_0(x)\,dx \\ \vdots \\ \int_a^b f(x)\varphi_n(x)\,dx \end{pmatrix}. \tag{1.9}$$

Pour notre problème avec C.L. de Dirichlet en a, cf (1.7) où $i = 1, ..., n$, on pose :

$$\left\{ \begin{array}{l} \bar{R} = [R_{ij}]_{\substack{1\le i\le n \\ 1\le j\le n}}, \qquad \bar{M} = [M_{ij}]_{\substack{1\le i\le n \\ 1\le j\le n}}, \\ \bar{R}_0 = [R_{i0}]_{1\le i\le n} \quad \text{(1ère colonne de } \bar{R}), \qquad \bar{M}_0 = [M_{i0}]_{1\le i\le n} \quad \text{(1ère colonne de } \bar{M}), \\ \bar{\vec{c}} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \quad \text{(l'inconnue du problème)}, \qquad \bar{\vec{f}} = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}. \end{array} \right.$$

Avec ces notations, le problème (1.7) s'écrit :

$$\left\{ \begin{array}{l} \text{Trouver } \bar{\vec{c}} \in \mathbb{R}^n : \\ (\bar{R} + \bar{M}).\bar{\vec{c}} = \bar{\vec{f}} - c_0(\bar{R}_0 + \bar{M}_0), \end{array} \right. \tag{1.10}$$

système résolu par exemple avec une méthode de type LU.

Conclusion : On vient de trouver $\vec{c}$. On reconstitue donc la fonction cherchée :

$$u_h(x) = \sum_{i=0}^{n} c_j \varphi_j(x).$$

Il reste à dessiner cette fonction : tout logiciel usuel de calcul permet un tel dessin.

Remarque 1.2 D'un point de vue pratique, souvent on n'extrait pas $\bar{R}$ de R, mais on pose :

$$(CL) = \begin{pmatrix} 10^7 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & & & \\ 0 & 0 & \dots & 0 \end{pmatrix}, \quad \tilde{R} = R + (CL), \quad \tilde{M} = M + (CL), \quad \tilde{f} = \begin{pmatrix} 10^7 c_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix},$$

et on résout :

$$\left\{ \begin{array}{l} \text{Trouver } \vec{c} \in \mathbb{R}^{n+1} : \\ (\tilde{R} + \tilde{M}).\vec{c} = \tilde{\vec{f}} \end{array} \right.$$

On impose ainsi la valeur c_0 indirectement : le terme 10^7 étant très grand devant les termes de la première ligne de R et de M, la première ligne de ce problème permet de trouver c_0 avec une très bonne approximation. La programmation est ainsi simplifiée (peu dans le cas 1-D mais beaucoup plus dans les cas 2-D et 3-D par exemple).

Remarque 1.3 Une autre méthode très proche de celle de la remarque précédente consiste à poser :

$$\hat{R} = \begin{pmatrix} 10^7 & (0 \quad \dots \quad 0) \\ \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} & [R_{ij}]_{\substack{1\le i\le n \\ 1\le j\le n}} \end{pmatrix}, \quad \hat{M} = \begin{pmatrix} 10^7 & (0 \quad \dots \quad 0) \\ \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} & [M_{ij}]_{\substack{1\le i\le n \\ 1\le j\le n}} \end{pmatrix}, \quad \hat{f} = \begin{pmatrix} 10^7 c_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix},$$

et on résout :

$$\begin{cases} \text{Trouver } \vec{c} \in \mathbb{R}^{n+1} : \\ (\hat{R} + \hat{M}).\vec{c} = \hat{\vec{f}}. \end{cases}$$

La condition aux limites c_0 est alors calculée de manière exacte. Bien sûr, au lieu de 10^7 on peut prendre tout simplement 1, mais prendre 10^7 permet dans une méthode de résolution de type LU avec pivot d'avoir un grand pivot en première place.

2 La programmation

La programmation consiste en le calcul des matrices M de masse et R de rigidité, ainsi que du vecteur second membre $\vec{f}$, voir (1.8) et (1.9).

2.1 Les f_i

Il s'agit de calculer, voir (1.9), pour $0 \le i \le n$:

$$f_i = \int_a^b f(x)\varphi_i(x)\,dx = \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} f(x)\varphi_i(x)\,dx.$$

On note, pour une fonction g quelconque, et $I_k = [x_{k-1}, x_k]$ le k-ième intervalle :

$$S_k(g) = \int_{I_k} g(x)\,dx = \int_{x_{k-1}}^{x_k} g(x)\,dx. \tag{2.1}$$

Et on remarque que le support des fonctions φ_i est dans $I_i \bigcup I_{i+1}$. Donc pour i=0 et i=n il reste :

$$f_0 = \int_{x_0}^{x_1} f(x)\varphi_0(x)\,dx = S_1(f\varphi_0), \quad f_n = \int_{x_{n-1}}^{x_n} f(x)\varphi_n(x)\,dx = S_n(f\varphi_n).$$

Et, pour $1 \le i \le n-1$:

$$f_i = \int_{x_{i-1}}^{x_i} f(x)\varphi_i(x)\,dx + \int_{x_i}^{x_{i+1}} f(x)\varphi_i(x)\,dx = S_i(f\varphi_i) + S_{i+1}(f\varphi_i).$$

2.1.1 Programmation "naturelle"

L'algorithme est élémentaire :

1- les extrémités sont calculées à part :
`f(0)=` $S_1(f\varphi_0)$ `et f(n)=` $S_n(f\varphi_n)$`.`

2- les f_i pour $i = 1, ..., n-1$:

```
for i=1 à n-1,     % pour chaque point intérieur,
   f(i)= S_i(fφ_i) + S_{i+1}(fφ_i).
end
```

Cette programmation élémentaire pose problème dans le cas d'éléments finis dans le plan (cas 2-D) ou dans l'espace (cas 3-D). Regardons pour simplifier le cas 2-D avec un maillage triangulaire : les fonctions chapeau de base $\varphi_i \in P_1$ sont définies comme étant continues sur $\mathbb{R}^2$, affines sur chaque triangle, et valant 1 au noeud i, et 0 aux autres noeuds (les noeuds étant par définition les sommets des triangles).

Et donc les fonctions chapeau ont leur support qui est l'union des triangles dont un sommet est le noeud i. La boucle précédente : "`for i=1:n-1`" n'est alors pas facile à programmer : il faut commencer, pour chaque noeud i à répertorier les triangles adjacents, avant de faire les calculs d'intégrale.

Une méthode plus simple est donnée au paragraphe suivant.

2.1.2 Programmation en vue du cas n-D

Sur chaque intervalles I_k on doit calculer les intégrales $S_k(f\varphi_i)$, mais uniquement pour $i = k$ ou $i = k + 1$, les autres cas donnant des intégrales nulles.

Ainsi pour le premier intervalle $k = 1$, on doit calculer :
1- $S_1(f, \varphi_0)$ (pour avoir f_0),
2- $S_1(f, \varphi_1)$ (pour avoir la première partie de f_1).

Pour le second intervalle $k = 2$, on doit calculer :
1- $S_2(f, \varphi_1)$ (pour compléter f_1),
2- $S_2(f, \varphi_2)$ (pour avoir la première partie de f_2).

Et on continue, d'où l'algorithme :

```
f(0)=0 % initialisation à 0 de f_0
for k=1 à n,      % on parcoure les n intervalles
    f(k-1)  = f(k-1)  + S_k(f,φ_{k-1})    % seconde partie de f_{k-1}
    f(k)    =           S_{k-1}(f,φ_k)    % première partie de f_k
end
```

Notez qu'avec cette programmation, les termes de bord f_0 et f_n se traitent de la même manière que les termes intérieurs (on n'a pas eu à les programmer à part).

Noter qu'en vue du cas n-D et des éléments finis P_ℓ pour $\ell \geq 2$, on programme en fait :

```
f=0 % initialisation à 0 de f⃗
for k=1 à n,      % on parcoure les n intervalles
    f(k-1)  = f(k-1)  + S_k(f,φ_{k-1})    % participation à f_{k-1}
    f(k)    = f(k)    + S_{k-1}(f,φ_k)    % participation à f_k
end
```

2.2 Les M_{ij} et R_{ij}

Il s'agit de calculer, voir (1.9) :

$$R_{ij} = \int_a^b p(x)\varphi_j'(x)\varphi_i'(x)\,dx \quad \text{et} \quad M_{ij} = \int_a^b q(x)\varphi_j(x)\varphi_i(x)\,dx.$$

Regardons par exemple M_{ij} (R_{ij} se traitant de la même manière).

On a :

$$M_{ij} = \sum_{k=1}^{n} \int_{I_k} q(x)\varphi_j(x)\varphi_i(x)\,dx = \sum_{k=1}^{n} T_k(i,j),$$

où on a posé $T_k(i,j) = \int_{I_k} q(x)\varphi_j(x)\varphi_i(x)\,dx$.

Ayant le support de φ_ℓ inclu dans $[x_{\ell-1}, x_{\ell+1}] = I_\ell \bigcup I_{\ell+1}$, il vient :

$$M_{ij} = \int_{(I_i \bigcup I_{i+1}) \bigcap (I_j \bigcup I_{j+1})} q(x)\varphi_j(x)\varphi_i(x)\,dx.$$

D'où :

$$\begin{cases} M_{ii} = T_i(i,i) + T_{i+1}(i,i). \\ M_{i,i-1} = T_i(i, i-1) \\ M_{i,i+1} = T_{i+1}(i, i+1), \\ j \notin \{i-1, i, i+1\} : M_{ij} = 0, \end{cases}$$

Comme dans le paragraphe 2.1.2, il est judicieux de faire une boucle sur les intervalles (surtout en vue d'une programmation en 2D ou en 3D) : ainsi pour le premier intervalle $k = 1$, on calcule :
1- $T_1(0,0)$ pour avoir M_{00},
2- $T_1(1,0)$ pour avoir M_{10}.
3- $T_1(0,1)$ pour avoir M_{01}.
4- $T_1(1,1)$ pour avoir la première partie de M_{11}.

Pour le second intervalle $k = 2$, on calcule :
1- $T_2(1,1)$ pour compléter M_{11},
2- $T_2(2,1)$ pour avoir M_{21}.
3- $T_2(1,2)$ pour avoir M_{12}.
4- $T_2(2,2)$ pour avoir la première partie de M_{22}.

Et on continue, d'où l'algorithme :

```
M=0 % initialisation de 0 de la matrice M
for k=1 à n,     % on parcoure les n intervalles
    M_{k-1,k-1}  = M_{k-1,k-1}  + T_k(k-1,k-1)    % seconde partie de M_{k-1,k-1}
    M_{k,k-1}    =                T_k(k,k-1)      % calcul de M_{k,k-1}
    M_{k-1,k}    =                T_k(k-1,k)      % calcul de M_{k-1,k}
    M_{k,k}      = M_{k,k}       + T_k(k,k)       % première partie de M_{k,k}
end
```

Notez que la matrice M est symétrique, donc on a $M_{k,k-1} = M_{k-1,k}$, et il est suffisant de calculer l'un de ces deux termes pourra voir l'autre sans calcul. En 2-D cependant, il n'y a aucune raison pour que la matrice M ou la matrice R soient symétriques (on demande seulement que $(R + M)$ soit coercitive, voir le cours d'éléments finis).

Noter qu'en vue du cas n-D et des éléments finis P_ℓ pour $\ell \geq 2$, on programme en fait :

```
M=0 % initialisation de 0 de la matrice M
for k=1 à n,     % on parcoure les n intervalles
    M_{k-1,k-1}  = M_{k-1,k-1}  + T_k(k-1,k-1)    % participation à M_{k-1,k-1}
    M_{k,k-1}    = M_{k,k-1}    + T_k(k,k-1)      % participation à M_{k,k-1}
    M_{k-1,k}    = M_{k-1,k}    + T_k(k-1,k)      % participation à M_{k-1,k}
    M_{k,k}      = M_{k,k}      + T_k(k,k)        % participation à M_{k,k}
end
```

Et cette structure de programme est conservée pour des éléments finis P_ℓ, cas $\ell \geq 2$, en n-D.

2.3 Remarque sur les calculs d'intégrales

Il est judicieux d'écrire son propre petit programme de calcul d'intégrale, par exemple à l'aide de la méthode de simpson :

$$\int_\alpha^\beta g(x)\,dx \simeq (\beta-\alpha)\frac{g(\alpha)+4g(\frac{\alpha+\beta}{2})+g(\beta)}{6}.$$

Cette formule est exacte pour tout g polynôme de degré inférieur ou égal à 3, et est en $O((\beta-\alpha)^4)$ pour toute fonction C^4. Comme pour nous les intervalles d'intégration sont les $]\alpha,\beta[=]x_{k-1},x_k[$ de longueur $h = x_k - x_{k-1}$ petite, cette formule donne de bons résultats.

En fait, pour des éléments finis P_1, qui donnent des résultats de précision $O(h)$, il suffit de considérer la méthode d'intégration des trapèzes :

$$\int_\alpha^\beta g(x)\,dx \simeq (\beta-\alpha)\frac{g(\alpha)+g(\beta)}{2},$$

qui est exacte pour les polynômes de degré ≤ 1 et en $O((\beta-\alpha)^2) = O(h^2)$ pour les fonctions C^2, et comme on doit faire la somme $\sum_{k=1}^n$ de termes en $O(h^2)$ avec $n = \frac{1}{h}$, la précision obtenue est en $O(h)$. Cependant, la méthode de Simpson n'est guère plus coûteuse (au moins en 1-D), et permet de considérer l'intégration des éléments finis P_3.

2.4 Exercice

Exercice 2.1 Vérifier que les fonctions chapeau définies § 1.2.1 sont dans $H^1([a,b])$.

Réponse. Regardons par exemple la fonction φ_3 (un "chapeau").

Cette fonction φ_3 est continue sur $[a,b]$, donc de carré continue, donc de carré intégrable. La fonction φ_3 par contre n'est pas dérivable au sens classique. Mais elle l'est au sens des distributions :

on a, pour toute fonction $\psi \in C^\infty([a,b])$, notant T_{φ_3} la distribution régulière associée à φ_3 :

$$\langle (T_{\varphi_3})', \psi \rangle \overset{\text{déf}}{=} -\langle \varphi_3, \psi' \rangle = -\int_a^b \varphi_3(x)\psi'(x)\,dx = -\int_{x_2}^{x_3} \varphi_3(x)\psi'(x)\,dx - \int_{x_3}^{x_4} \varphi_3(x)\psi'(x)\,dx,$$

puisque φ_3 est nulle sur $[x_0, x_2] \bigcup [x_4, x_n]$. Et φ_3 est affine sur $[x_2, x_3]$ et sur $[x_3, x_4]$, de dérivée valant respectivement $\frac{1}{x_3 - x_2}$ et $-\frac{1}{x_4 - x_3}$. Donc, après intégration par parties :

$$\begin{aligned} \langle (T_{\varphi_3})', \psi \rangle &= \int_{x_2}^{x_3} \frac{1}{x_3 - x_2}\psi(x)\,dx + \int_{x_3}^{x_4} -\frac{1}{x_4 - x_3}\psi(x)\,dx + 0 \\ &= \int_a^b (\frac{1}{x_3 - x_2} 1_{]x_2, x_3[} - \frac{1}{x_4 - x_3} 1_{]x_3, x_4[})\psi(x)\,dx. \end{aligned}$$

Et donc $(T_{\varphi_3})'$ est la distribution régulière associée à la fonction $\frac{1}{x_3 - x_2} 1_{]x_2, x_3[} - \frac{1}{x_4 - x_3} 1_{]x_3, x_4[}$ (c'est la dérivée usuelle là où elle pouvait être calculée, i.e. ailleurs qu'aux coins x_2, x_3 et x_4). On la note simplement $(T_{\varphi_3})' = \varphi_3'$.

Et la fonction en escalier φ_3' est bien de carré intégrable (elle est définie en escalier presque partout donc intégrable pour la mesure de Lebesgue).

On fait les mêmes calculs pour toutes les fonctions φ_i pour $i = 0, ..., n$, les fonctions demi-chapeau φ_0 et φ_n ne posant pas de problème particulier (il n'y a qu'un terme à intégrer par parties au lieu de deux), puis toute fonction de P_1 étant combinaison linéaire des φ_i, toute fonction de P_1 est également dans $H^1([a,b])$. ∎

```
%%%%%%%%%%%%%%   programme principal progp1.m   %%%%%%%%%%%%%%%%%%
% programme de calcul de la solution du probleme du Laplacien
%    en 1-D avec elts finis P1.
%
% Résolution sur l'intervalle [aa,bb] de :
%    -beta u'' + alpha u = f              (*)
%    pour les CL de Dirichlet homogenes;

beta = 1, alpha = 1, % coefficient de l'équation (*) à résoudre
% fonction f dans (*) ; f = 'fb', voir fin

aa = 0, bb = pi, if aa>=bb, error('aa doit etre inferieur a bb'),end
n = 10, h = (bb-aa)/n, % nombre d'intervalles et pas uniforme de chaque intervalle
xm = aa:h:bb % les points du maillage : xm(1)=aa, xm(2)=aa+h, ..., xm(n+1)=bb.
%xm = linspace(aa,bb,n+1) % les points du maillage : xm(1)=aa, xm(2)=aa+h, ..., xm(n+1)=bb.

% Assemblage du membre de droite :
F = sparse(n+1,1); % initialisation
for k = 1:n   % Considération du k-ieme intervalle [xm(k),xm(k+1)] :
  for il = 1:2,       % il = ilocal
     ig = k-1 + il;   % ig = iglobal
     F(ig) = F(ig) + integsimpson('phibasef',xm(k) ,xm(k+1),il,k,xm);
   end
end

% Assemblage matrices :
K = sparse(n+1,n+1); % initialisation
M = K; % initialisation
for k = 1:n   % Considération du k-ieme intervalle [xm(k),xm(k+1)] :
  for il =1:2,       % il = ilocal
    ig = k-1 + il;   % ig = iglobal
    for jl = 1:2,    % jl = jlocal
      jg = k-1 + jl; % jg = jglobal
      M(ig,jg) = M(ig,jg) + integsimpson('phiphibase',xm(k),xm(k+1),il,jl,k,xm);
      K(ig,jg) = K(ig,jg) + integsimpson('phiderphiderbase',xm(k),xm(k+1),il,jl,k,xm);
      end
   end
end
K = beta*K + alpha*M; % Matrice du membre de gauche :

% Prise en compte des conditions de Dirichlet homogène, par pénalisation :
K(1,1) = 10^7; K(n+1,n+1) = 10^7;
F(1) = 0; F(n+1) = 0; % plus généralement F(1) = u(aa) et F(n+1) = u(bb) données du problème

% Prise en compte des conditions de Dirichlet homogène, de manière exacte :
%   sprintf('conditions aux limites de Dirichlet')
%   K=K([2:n-1,2:n-1]); M=M([[2:n-1,2:n-1]]); F=F([2:n-1]);
% et dans la solution on doit faire :
%   u(2:n)=u;u(1)=0;u(n+1)=0;  % plus généralement u(1) = u(aa) et u(n+1) = u(bb)

% Calcul de la solution :
u = K\F; % résolution de K*u = F par la méthode LU.

plot(xm,u) % Dessin du graphe de la solution u(x) :

%%%%%%%%%%%%%%%%%%%    Les fonctions utilisées   %%%%%%%%%%%%%%%%%%%

function y = phibase(x,il,k,xm)
% Pour il=1,2, les 2 fonctions phi_{il} de base P1 sur le kième intervalle [xm(k),xm(k+1)]
%    (les deux demi-chapeaux). Ici il = numéro local à l'intervalle.

if (il<1 | il>2),error('il y a exactement deux fonctions de base sur un intervalle'),end
if (il==1)
   y = (x-xm(k+1)) ./ (xm(k)-xm(k+1)); % décroissante sur [xm(k),xm(k+1)]
```

```
else
   y = (x-xm(k)) ./ (xm(k+1)-xm(k)); % croissante sur [xm(k),xm(k+1)]
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = phiphibase(x,il,jl,k,xm)
% produit phi_{il}*phi_{jl} des fonctions de base P1 sur le k-ième intervalle [xm(k),xm(k+1)] :

y = phibase(x,il,k,xm) .* phibase(x,jl,k,xm);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = phiderbase(x,il,k,xm)
% Pour il=1,2, les derivees phi_{il}' sur le k-ième intervalle [xm(k),xm(k+1)]
%  Voir les fonctions de base  pbaseab.m

if (il<1 | il>2),error('il n''y a que deux fonctions de base'),end
if (il==1)
   y = -ones(size(x)) / (xm(k+1)-xm(k)); % ou bien pdbaseab = -x.^0
else
   y = ones(size(x)) / (xm(k+1)-xm(k));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = phiderphiderbase(x,il,jl,k,xm)
% produit phi_{il}'*phi_{jl}' des dérivées des fonctions de base, sur le k-ième intervalle

y = phiderbase(x,il,k,xm).*phiderbase(x,jl,k,xm);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = phibasef(x,il,k,xm)
% produit phi_{il}(x)*f(x) sur l'intervalle k, sur le k-ième intervalle

y = phibase(x,il,k, xm).*fb(x);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = fb(x)
% fonction pour le membre de droite

y = 10*x.^0; %y=10*ones(size(x));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function val = integsimpson(fun,a,b,varargin)
% intégration de fonction fun par simpson sur [a,b]

c = (a+b) / 2;
y = feval(fun,[a c b],varargin{:});
fa = y(1);
fc = y(2);
fb = y(3);
val = (b-a) * (fa+4*fc+fb) / 6;
```