TP 2 Concepts mathématiques pour la CAO : Analyse isogéometrique

CONSTANZA Corentin

2020

1 Formulation faible du problème

On considère le problème de trouver $u \in C^2(]a, b[)$ satisfaisant

$$\begin{cases} -\dfrac{\partial^2 u}{\partial x^2}(x) = f(x),\ x \in]0, 1[\\ u(a) = \alpha,\ u(b) = \beta \end{cases} \tag{1}$$

1. Ecrire ce problème sous une formulation faible. La formulation faible du problème est la suivante : On cherche u dans $V_u = \{u \in L^2(\Omega), u' \in L^2(\Omega), u(0) = \alpha, u(1) = \beta\}$ et on prend v dans $V = \{v \in L^2(\Omega), v' \in L^2(\Omega), u(0) = 0, u(1) = 0\}$ tel que

   $\int_\Omega u'(x)v'(x)dx = \int_\Omega f(x)v(x)$

2. On souhaite représenter la solution $\bar{u}(\xi)$ dans une base de NURBS associée aux vecteurs de noeuds

   $$U \quad = \quad \{\xi_0 = 0, 0, 0, 0, 1/4, 1/2, 3/4, 1, 1, 1, 1 = \xi_m\} \tag{2}$$

   Quel est la dimension de l'espace des fonctions de base NURBS ?

   La dimension de l'espace de base des NURBS est 2.

3. Décrire les fonctions de base NURBS qui rentrent dans l'espace des fonctions poids de la formulation faible. On utilise pour cela la formule de recurence pour $p >= 1$:

   $N_i^p(x) = \frac{x - x_i}{x_{i+p} - x_i} N_i^{p-1}(x) + \frac{x_{i+p+1} - x}{x_{i+p+1} - x_{i+1}} N_{i+1}^{p-1}(x)$

   On obtient le tableau suivant :

$U = \{\xi_0 = 0, 0, 0, 0, 1/4, 1/2, 3/4, 1, 1, 1, 1 = \xi_{10}\}$

	$[0, 1/4[$	$[1/4, 1/2[$	$[1/2, 3/4[$	$[3/4, 1[$
$N_{0,0}$	0	0	0	0
$N_{1,0}$	0	0	0	0
$N_{2,0}$	0	0	0	0
$N_{3,0}$	1	0	0	0
$N_{4,0}$	0	1	0	0
$N_{5,0}$	0	0	1	0
$N_{6,0}$	0	0	0	1
$N_{7,0}$	0	0	0	0
$N_{0,1}$	0	0	0	0
$N_{1,1}$	0	0	0	0
$N_{2,1}$	$1-4\xi$	0	0	0
$N_{3,1}$	4ξ	$2-4\xi$	0	0
$N_{4,1}$	0	$4\xi-1$	$3-4\xi$	0
$N_{5,1}$	0	0	$4\xi-2$	$4-4\xi$
$N_{6,1}$	0	0	0	$4\xi-3$
$N_{7,1}$	0	0	0	0
$N_{0,2}$	0	0	0	0
$N_{1,2}$	$(1-4\xi)^2$	0	0	0
$N'_{1,2}$	$-8(1-4\xi)$	0	0	0
$N_{2,2}$	$8\xi(1-3\xi)$	$2(1-2\xi)^2$	0	0
$N'_{2,2}$	$8-48\xi$	$-8(1-2\xi)$	0	0
$N_{3,2}$	$8\xi^2$	$-16\xi^2+12\xi-3/2$	$1/2(3-4\xi)^2$	0
$N'_{3,2}$	16ξ	$-32\xi+12$	$-4(3-4\xi)$	0
$N_{4,2}$	0	$1/2(4\xi-1)^2$	$-16\xi^2+20\xi-11/2$	$8(1-\xi)^2$
$N'_{4,2}$	0	$4(4\xi-1)$	$-32\xi+20$	$-16(1-\xi)$
$N_{5,2}$	0	0	$2(2\xi-1)^2$	$8(1-\xi)(3\xi-2)$
$N'_{5,2}$	0	0	$8(2\xi-1)$	$-48\xi+40$
$N_{6,2}$	0	0	0	$(4\xi-3)^2$
$N'_{6,2}$	0	0	0	$8(4\xi-3)$
$N_{7,2}$	0	0	0	0

4. Ecrire formellement la matrice de rigidité locale et le vecteur force local correspondant à la formulation faible du problème en ayant une représentation NURBS de la solution associé à un élément.

   La matrice de rigidité locale s'écrit :

$$a_k(N_{i,p}, N_{j,p}) = \sum_{k=0}^{n} \int_{x_k}^{x_{k+1}} \frac{\partial N_{i,p}}{\partial x}(x) \frac{\partial N_{j,p}}{\partial x}(x) dx \tag{3}$$

5. Ecrire formellement la matrice de rigidité globale et le vecteur de force global associé au problème. La matrice de rigidité globale s'écrit :

$$a_k(N_{i,p}, N_{j,p}) = \sum_{k=0}^{n} \int_{x_k}^{x_{k+1}} \frac{\partial N_{i,p}}{\partial x}(x) \frac{\partial N_{j,p}}{\partial x}(x) dx \tag{4}$$

   De plus nous avons pour tout $j \in [1, n]$:

$$b(N_{j,p}(x)) = a(\alpha N_{0,p}(x) + \sum_{i=1}^{n} u_i N_{i,p}(x) + \beta N_{n+1,p}(x), N_{j,p}(x)) \tag{5}$$

Or comme a est bilinéaire on peut écrire :

$$b(N_{j,p}(x)) = \alpha a(N_{0,p}(x), N_{j,p}(x)) + \sum_{i=1}^{n} u_i a(N_{i,p}(x), N_{j,p}(x)) + \beta a(N_{n+1,p}(x), N_{j,p}(x)) \quad (6)$$

ce qui donne :

$$\sum_{i=1}^{n} u_i a(N_{i,p}(x), N_{j,p}(x)) = b(N_{j,p}(x)) - \alpha a(N_{0,p}(x), N_{j,p}(x)) - \beta a(N_{n+1,p}(x), N_{j,p}(x)) \quad (7)$$

2 Construction du système

Pour construire ce système, nous aurons besoins de differentes fonction que nous allons detaillé ci dessous ainsi que les fonction Dichotomie(FindSpan) et BasisFuns du TP précedent. La fonction IEN est un tableau de taille (munique-1,p+1) (dans notre cas munique-1 = p+1) qui à chaque ligne nous renvoie le numéro des NURBS non nulles sur l'intervale $[x_k; x_{k+1}]$. Elle s'écrit comme ceci :

```
function [IEN] = IEN(p,munique)
%dans notre cas particulier, munique-1=p+1 on a donc une matrice carr

IEN=zeros(munique-1,p+1);
for k=1:munique-1
    if (k==1)

        for i=2:p+1
            IEN(k,i)=i-1;
        end

    elseif (k==munique-1)

        for i=1:p
            IEN(k,i)=p+i-1;
        end
    else
        for i=1:p+1
            IEN(k,i)=k-2+i;
        end
    end
end
end
```

Nous aurons besoin de la fonction dBasisFuns qui calcule les $\frac{\partial N_l}{\partial \xi}(\xi)$. Elle fonctionne sur le même principe que la fonction BasisFuns du TP 1. Elle s'écrit comme ceci :

```
function [N] = dBasisFuns(u,p,U)
k=Dichotomie(p,u,U);
left=zeros(p+1,1);
right=zeros(p+1,1);
N=zeros(1,p+1);
N(1)=1;
for j=1:p
    left(j+1)=u-U(k+1-j);
    right(j+1)=U(k+j)-u;
    saved=0;
    if j<p
        for r=0:j-1
            temp=N(r+1)/(right(r+2)+left(j-r+1));
            N(r+1)=saved + right(r+2)*temp;
            saved=left(j-r+1)*temp;
        end
    else
```

```
        for r=0:j-1
            temp=N(r+1)/(right(r+2)+left(p-r+1));
            N(r+1)=saved - p*temp;
            saved=+p*temp;
        end
    end
    N(j+1)=saved;
end
N(p+1)=saved;
end
```

La fonction "a" qui nous calcule grâce à la méthode de Simpson les intégrales de la forme :

$$a^{e_i}(N_l, N_j) \quad = \quad \int_{\xi_i}^{\xi_{i+1}} \frac{\partial N_l}{\partial \xi}(\xi)\frac{\partial N_j}{\partial \xi}(\xi)d\xi \tag{8}$$

Cette fonction s'écrit de la facon suivante :

```
function A = a(k,i,j,p,U,V)
%calcul de a par la formule de Simpson

a=V(k);
b=V(k+1);

%init des differents coefficients utilises dans la formule de Simpson
dNa=dBasisFuns(a,p,U);
dNb=dBasisFuns(b,p,U);
dNab=dBasisFuns((a+b)/2,p,U);

%formule de Simpson
A=((b-a)/6)*(dNa(i)*dNa(j)+ 4*(dNab(i)*dNab(j)) + dNb(i)*dNb(j));
end
```

La fonction MatLoc qui nous permets de construire nos matrices de rigidités locales à partir des indices que l'on extrait de IEN. Elle s'écrit comme ceci :

```
function Ml = MatLoc(k,p,U)
%Construction des Matrices locales
%Init
[V,munique,~] = Vunique(U);
Ml=zeros(munique,munique);

%extraction des indices des Nurbs non nul de IEN
I=IEN(p,munique);
I=I(k,:);

%calcul des coef de la matrices local
for i=1:length(I)
    for j=1:length(I)
        if (I(i)~=0) && (I(j)~=0)
            Ml(I(i),I(j))=a(k,i,j,p,U,V);
        end
    end
end
end
```

La fonction MatGlob quand elle nous permet de crée la matrice de rigidité globale notre système en assamblant les differentes matrices locales. Elle s'écrit comme ci-dessous :

```
function M = MatGlob(p,U,munique)
%Assemblage de la matrice globale
%Init de M
M=zeros(munique,munique);

for k=1:munique-1
```

```

    %calcul des Matrice local
    Ml=MatLoc(k,p,U);

    %assemblage
    M = M + Ml;
end
end
```

Nous allons maintenant passer à la construction du second membre. Pour ce faire nous aurons besoin de deux fonction la première "F" sera exactement la même fonction que "a", nous remplaceront juste dBasisFuns par BasisFuns. La seconde quand à elle est la fonction SecondMembre qui s'écrit comme ceci :

```
function SM = SecondMembre(f,alpha,beta,p,U,munique)
%calcul du second membre

Iinterval=IEN(p,munique);
[V,munique,~] = Vunique(U);
b=zeros(munique,1);

%calcul des coeff
for k=1:munique-1
    I=Iinterval(k,:);

    %calcul des coeff generaux
    for i=1:length(Iinterval)
        if (I(i)~=0)
            SM(I(i)) = SM(I(i)) + F(f,k,i,p);
        end
    end

    %retrait des coeff dependant de alpha et beta
    for i=max(k-p-1,1):k
        SM(i)=SM(i)-alpha*a(1,i,1,p,U,V);
        SM(i)=SM(i)-beta*a(4,i,4,p,U,V);
    end
end
end
```

3 Résolution et test du système

Nous allons maintenant vérifier la bonne implementation de notre programme à l'aide d'une solution manufacturée. Nous prendrons $u_exa(x) = -x*(x-1)$ et donc $f(x) = 2$. Nous allons d'abord construire une fonction solution qui nous permettra de créer notre solution approchée composante par composante. (car nous utiliserons une boucle for qui parcourera l'intervale discrétisé dasn la focntion suivante) Elle s'écrit comme ceci :

```
function u = solution(t,f,alpha,beta,p,U,V,munique)
%calcul de la solution approche
%Init
Usol=MatGlob(p,U,munique)\SecondMembre(f,alpha,beta,p,U,munique);
N=BasisFuns(t,p,U);
u=0;
%test pour savoir dans quel interval on se situe afin de ne pas calculer
%les differentes nurbs non nul.
if t < V(2)
        u = u + alpha*N(1);
        for i=2:4
            u = u + Usol(i-1)*N(i);
        end
elseif t < V(3)
        for i=1:4
```

```
        u = u + Usol(i)*N(i);
        end
elseif t < V(4)
        for i=1:4
        u = u + Usol(i+1)*N(i);
        end
elseif t <V(5)
        u = u + beta*N(4);
        for i=1:3
            u = u + Usol(i+2)*N(i);
        end
end
end
```

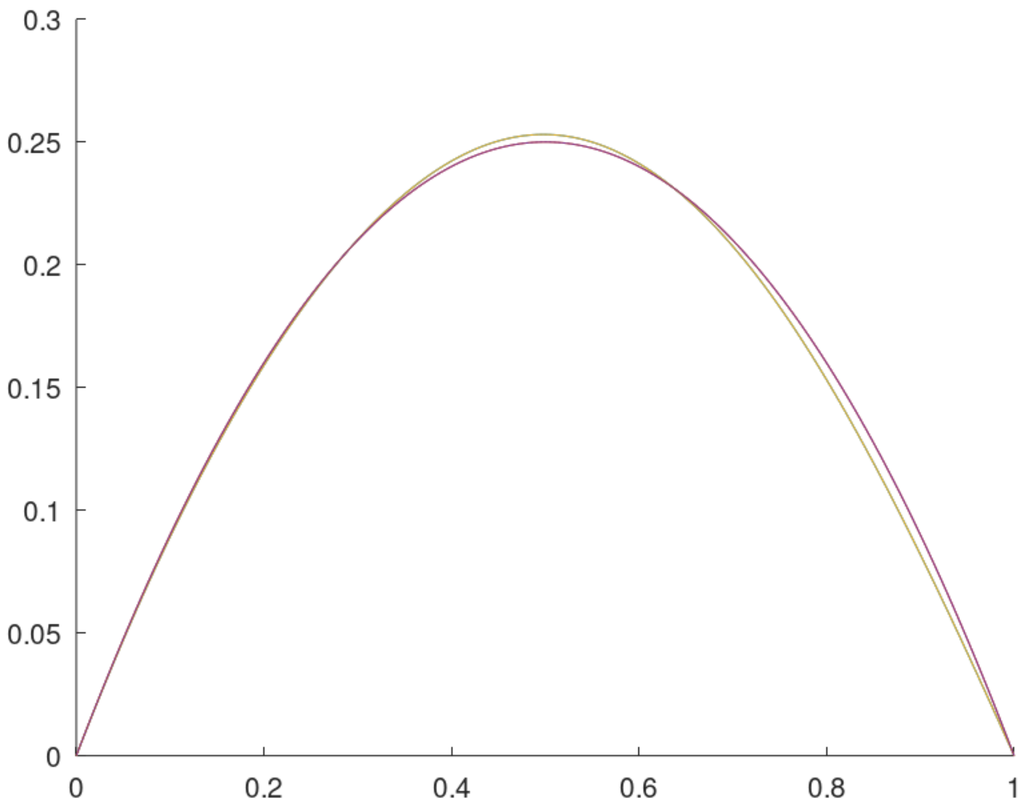
La fonction suivante, nommé Solutionmanufact nous permet de crée notre solution exacte, notre solution approché (en appelant la fonction précédante) et de tracer les differentes courbes. Elle s'écrit comme ceci :

```
function [] = Solutionmanufact(p,f,alpha,beta,U,V,munique)
%Test de notre solution approche par la technique des solutions
%manufacturee

%Solution manufacturee
x=linspace(0,1,500);
uexa=-x.*(x-1);

%Discretisation
u=zeros(500,1);
W=linspace(V(1),V(5),500);
%calcul de la solution approche
for j=1:length(W)
    u(j)=solution(W(j),f,alpha,beta,p,U,V,munique);
end

%plot des courbes differentes courbes
hold on
plot(x,u)
plot(x,uexa)
end
```

Nous utiliserons le script suivant pour tester l'ensemble de nos fonctions

```
clc
clear
%init
U=[0,0,0,0,1/4,2/4,3/4,1,1,1,1];
[V,munique,S] = Vunique(U);
p=3;
f=@(x)(2);
alpha=0;
beta=0;
%test solution manufacturer
Solutionmanufact(p,f,alpha,beta,U,V,munique)
```

Et nous avons en sortie les courbes suivante :



Notre solution approche bien la solution exacte, notre programme semble donc bien implementée.