EPL	
Avril 2022	*Introduction aux éléments finis*
LEPL1110	**Solution**

1 Pour un triangle linéaire parent définis par les trois sommets $[0,0],[1,0],[1,1]$, calculer la matrice locale définie par :

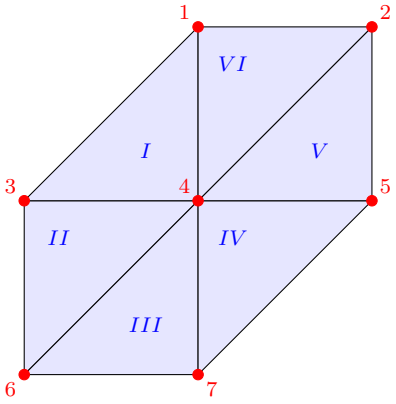
$$\widehat{A}_{ij} = \int_{\widehat{\Omega}} \frac{\partial \phi_i}{\partial \xi}\frac{\partial \phi_j}{\partial \xi} + \frac{\partial \phi_i}{\partial \eta}\frac{\partial \phi_j}{\partial \eta}\, d\xi d\eta \;=\; \frac{1}{2}\begin{array}{|c|c|c|}\hline 1 & -1 & 0 \\ \hline -1 & 2 & -1 \\ \hline 0 & -1 & 1 \\ \hline \end{array}$$

2 On considère maintenant un maillage composé de six triangles...

	1	2	3
I	3	4	1
II	6	4	3
III	6	7	4
IV	7	5	4
V	4	5	2
VI	4	2	1

	X_i	Y_i
1	1.0	2.0
2	2.0	2.0
3	0.0	1.0
4	1.0	1.0
5	2.0	1.0
6	0.0	0.0
7	1.0	0.0

Dessiner le maillage en y indiquant noeuds et éléments :-)



En assemblant les six matrices locales, obtenir la matrice globale définie par :

$$A_{ij} = \int_{\Omega} \frac{\partial \tau_i}{\partial x}\frac{\partial \tau_j}{\partial x} + \frac{\partial \tau_i}{\partial y}\frac{\partial \tau_j}{\partial y}\, dxdy \;=\; \frac{1}{2}\begin{array}{|c|c|c|c|c|c|c|}\hline 3 & -1 & & -2 & & & \\ \hline -1 & 2 & & & -1 & & \\ \hline & & 3 & -2 & & -1 & \\ \hline -2 & & -2 & 8 & -2 & & -2 \\ \hline & -1 & & -2 & 3 & & \\ \hline & & -1 & & & 2 & -1 \\ \hline & & & -2 & & -1 & 3 \\ \hline \end{array}$$

On peut immédiatement observer que les matrices locales $A^I_{ij} = A^{III}_{ij} = A^V_{ij} = \widehat{A}_{ij}$.
Les autres matrices locales $A^{II}_{ij} = A^{IV}_{ij} = A^{VI}_{ij}$ sont aussi égales entre elles, mais légèrement différentes de $\widehat{A}_{ij}$.
Toutefois, elles peuvent être déduites à partir de $\widehat{A}_{ij}$ en effectuant judicieusement quelques permutations.

Prière de remplir, en MAJUSCULES, votre nom, votre prénom, votre noma et votre numéro magique sur chaque face.

LEPL1110 **Avril 2022**	Nom :	**Numéro magique**
	Prénom :	**Noma**

3

La fonction suivante effectue l'intégration numérique d'une fonction quelconque par une règle de Gauss-Legendre à 4 points sur un quadrilatère défini par ses quatre sommets. Malheureusement, quelques lignes de code ont été subtilisées par un informaticien facétieux. Il s'agit donc de compléter l'implémentation ci-dessous.

```
double gaussIntegrate(double x[4], double y[4], double (*f) (double, double))
{
    double I = 0;
    const double a = 0.577350269189626;
    const double xsi[4]    = {  a,  a, -a, -a};
    const double eta[4]    = {  a, -a, -a,  a};
    const double weight[4] = {1.0,1.0,1.0,1.0};
    int i,j;

    for (i=0; i<4; i++) {
        double xsiLoc = xsi[i];
        double etaLoc = eta[i];
        double phi[4] = { (1+xsiLoc)*(1+etaLoc)/4.0, (1-xsiLoc)*(1+etaLoc)/4.0,
                          (1-xsiLoc)*(1-etaLoc)/4.0, (1+xsiLoc)*(1-etaLoc)/4.0} ;
        double dphidxsi[4] = {(1+etaLoc)/4.0, -(1+etaLoc)/4.0, -(1-etaLoc)/4.0,  (1-etaLoc)/4.0};
        double dphideta[4] = {(1+xsiLoc)/4.0,  (1-xsiLoc)/4.0, -(1-xsiLoc)/4.0, -(1+xsiLoc)/4.0};
        double dxdxsi = 0;
        double dydxsi = 0;
        double dxdeta = 0;
        double dydeta = 0;

        double xLoc = 0;
        double yLoc = 0;
        for (i=0; i<4; i++)
        {
            xLoc += x[i]*phi[i];
            yLoc += y[i]*phi[i];
            dxdxsi += x[i]*dphidxsi[i];
            dydxsi += y[i]*dphidxsi[i];
            dxdeta += x[i]*dphideta[i];
            dydeta += y[i]*dphideta[i];
        }
        double jacobian = dydeta*dxdxsi-dydxsi*dxdeta;
        I = I + f(xLoc,YLoc)*weigth[i]*jacobian;

     }
    return I;
}
```

4

L'ensemble des solutions de l'équation aux dérivées partielles

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} \quad = \quad 0$$

est donné par une des options ci-dessous :

1. $u(x,t) = f(x-ct)$
2. $u(x,t) = g(x+ct)$
3. $u(x,t) = f(x-ct) + g(x+ct)$
4. $u(x,t) = f(x-ct)\ g(x+ct)$

Indique ton choix ici :-)

1. : $u(x,t) = f(x-ct)$

où les fonctions f et g sont déterminées par les conditions aux limites.

5 Démontrer l'inégalité de Cauchy-Schwarz : $| < u, v > | \leq \|u\| \, \|v\|$

$\forall$ éléments u, v et $\forall$ réel non nul λ, on peut écrire :

$$0 \leq < u + \lambda v, u + \lambda v >$$

$\downarrow$ En vertu de la linéarité du produit scalaire !

$$0 \leq < u, u > + 2\lambda < u, v > + \lambda^2 < v, v >$$

Pour que ce polynôme du second degré en λ soit toujours positif, il faut que le réalisant soit négatif :

$$\begin{aligned} < u, v >^2 - < u, u >< v, v > &\leq 0 \\ < u, v >^2 &\leq < u, u >< v, v > \\ | < u, v > | &\leq \|u\| \, \|v\| \end{aligned}$$

□

6 Définir[1] $H_1(]0,1[)$: espace fonctionnel de Sobolev sur l'intervalle $]0,1[$:

$$H_1(]0,1[) = \left\{ v(x) :]0,1[\to \mathbb{R} \text{ tels que } \int_0^1 \Big(v(x)\Big)^2 + \Big(v'(x)\Big)^2 \, dx < \infty \right\}$$

7 Considérons un espace d'Hilbert quelconque.
Quelle est l'unique affirmation exacte ?

1. Tout espace vectoriel avec un produit scalaire est un espace d'Hilbert.
2. Tout espace d'Hilbert est un espace de Banach.
3. Un espace d'Hilbert ne contient que des formes continues et coercives.
4. Les espaces d'Hilbert sont des espaces fonctionnels.

Indique ton choix ici :-)

2. : Tout Hilbert est un Banach.

[1] La question précise bien que l'on se restreint au cas unidimensionnel : il ne faut donc pas me sortir des tas de dérivées partielles dans la définition : hein :-)

8

Avec des éléments finis linéaires continus, on calcule une solution approchée $u^h(x) \approx u(x)$ du problème de Cauchy

$$\begin{cases} u''(x) + f(x) &= 0 \qquad x \in]0,1[\\ u(0) &= 0 \\ u(1) &= 0 \end{cases}$$

Démontrer formellement que les valeurs nodales obtenues ainsi obtenues seront parfaitement exactes !
En d'autres mots $u^h(X_i) = u(X_i)$ pour les noeuds X_i du maillage.

La solution discrète $u^h(x) = \sum U_i \tau_i(x)$ est obtenue en résolvant le problème suivant :

$$\sum_j < \tau_i' \tau_j' > U_j \; = < \tau_i f >$$

Il faut juste démontrer que $u(X_i)$ satisfont exactment les mêmes équations :-)
On écrit donc :

$$u'' + f = 0$$

$$\downarrow$$

$$< \tau_i u'' > + < \tau_i f > = 0$$

En intégrant par parties sur chaque élément e !
Tous les termes frontières s'annulent par conservation !

$$- \sum_e < \tau_i' u' >_{\Omega_e} + \underbrace{\sum_e \left[\tau_i u' \right]_e}_{=0} + < \tau_i f > = 0$$

En intégrant par parties à nouveau sur chaque élément e !
Et en observant que $\tau_i'' = 0$ dans tout élément !

$$\underbrace{\sum_e < \tau_i'' u >_{\Omega_e}}_{=0} - \underbrace{\sum_e \left[\tau_i' u \right]_e}_{\frac{-u(X_{i-1})+2u(X_i)-u(X_{i+1})}{h}} + < \tau_i f > = 0$$

On obtient ainsi une expression de différences finies :-)
En remplaçant u par $\tilde{u}(x) = \sum u(X_i)\tau_i$!
Car l'expression obtenue ne dépend que des valeurs nodales !
Ce qui n'est vrai qu'en 1D !

$$\sum_e < \tau_i'' \tilde{u} >_{\Omega_e} - \sum_e \left[\tau_i' \tilde{u} \right]_e + < \tau_i f > = 0$$

En effectuant l'opération inverse de l'intégrale par parties...

$$- < \tau_i' \tilde{u}' > + < \tau_i f > = 0$$

Et en développant l'interpolation $\tilde{u}$!

$$\sum_j < \tau_i' \tau_j' > u(X_j) = < \tau_i f > \qquad \square$$

La clé du miracle : la frontière entre deux éléments 1D est un point unique associé à une unique valeur nodale !
Ce n'est évidemment plus le cas en 2D et 3D malheureusement !
La démonstration est faite avec des éléments de même taille h, mais ce n'est en rien une contrainte :-)

9 Pour modéliser notre tsunami en omettant le terme de Coriolis, nous considérons le modèle unidimensionnel :

$$\begin{cases} \dfrac{\partial \eta}{\partial t} + h\dfrac{\partial u}{\partial x} &= 0 \\ \dfrac{\partial u}{\partial t} + g\dfrac{\partial \eta}{\partial x} &= \dfrac{\tau}{\rho h} - \gamma u \end{cases}$$

Donner les unités de τ et de γ.
Expliquer brièvement le sens physique de ces deux paramètres.

Le terme γu représente la dissipation des échelles non-résolues. Les unités de γ sont $\left[\frac{1}{s}\right]$

Le terme $\frac{\tau}{\rho h}$ représente le forçage dû au vent. Les unités de τ sont celles d'une pression $\left[\frac{N}{m^2}\right]$

10 Nous allons maintenant ajouter le terme d'inertie à notre modèle : cela aura évidemment un impact sur la vitesse de propagation de l'onde qui propage à l'avant et à l'arrière de l'impact initial. La norme de deux vitesses ne sera plus identique !

$$\begin{cases} \dfrac{\partial \eta}{\partial t} + h\dfrac{\partial u}{\partial x} &= 0 \\ \dfrac{\partial u}{\partial t} + u\dfrac{\partial u}{\partial x} + g\dfrac{\partial \eta}{\partial x} &= \dfrac{\tau}{\rho h} - \gamma u \end{cases}$$

Calculer les deux vitesses de propagation en calculant les valeurs propres de la matrice du système différentiel.

On calcule les valeurs propres du problème linéarisé en écrivant :

$$0 = \det \begin{bmatrix} -\lambda & h \\ g & u-\lambda \end{bmatrix}$$

$$\downarrow$$

$$\lambda^2 - u\lambda - gh = 0$$

On conclut que $\lambda = \dfrac{u \pm \sqrt{u^2 + 4gh}}{2}$

11

Ecrire une fonction qui alloue une matrice de réels de taille $n \times n$, en imposant que l'ensemble des composantes de la matrice soient stockées de manière parfaitement contigue dans la mémoire de l'ordinateur. Ensuite, initialiser la matrice par l'expression $A_{ij} = ni + j$, en utilisant respectivement une expression du type `A[i][j]` ou du type `B[i]` en ayant une vue de la mémoire allouée comme un tableau à deux indices ou comme un vecteur à un indice.

```
#include <stdlib.h>

double** matrixCreate(int size)
{

    double **A = malloc(sizeof(double*) * size);
    A[0] = malloc(sizeof(double) * size * size);
    for (int i=1 ; i < size ; i++)
        A[i] = A[i-1] + size;
    return A;

}

int main() {
    int size = 4;
    double **A = matrixCreate(size);
    double *B = A[0];


    // Initialisation comme un vecteur

    for (int i=0 ; i < size*size ; i++)
        B[i] = i;

    // Initialisation équivalente comme un tableau
    // Evidemment, on refait ce qui vient d'avoir été fait !

    for (int i=0; i < size; i++)
        for (int j=0; j < size; j++)
            A[i][j] = i*size+j;

}
```

12

On compile avec `gcc` le programme suivant et si un exécutable est créé, on l'exécute.

```
#include <stdio.h>
int main() {
  int j, tab[3];
  int *ptr = tab;
  for(j=0; j < 3; j++)
    tab[j] = 5;
  *(ptr + 1) = 3;
  printf("[ %d %d %d ]\n", tab[0], tab[1], tab[2]);
  return 0;}
```

1. Il n'est pas possible de le compiler.
2. Il provoque une erreur à l'exécution.
3. Il fonctionne et affiche `[5,5,5]`
4. Il fonctionne et affiche `[3,5,5]`
5. Il fonctionne et affiche `[5,3,5]`

Indique ton choix ici :-)

5. : Il affiche `[5,3,5]`.