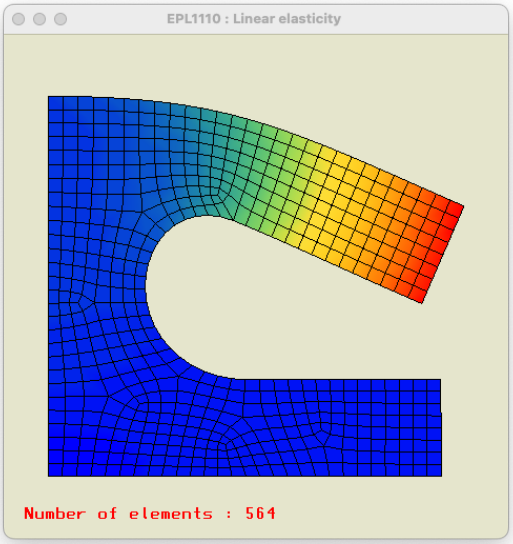
Finite elements for dummies : Elasticité linéaire !



Ce dernier devoir est consacré à l'implémentation des équations de l'élasticité linéaire pour des éléments quadrilatère bilinéaires ou des triangles linéaires.

Nous allons considérer un profil métallique en acier en forme de I qui se déforme sous son poids propre et tirer profit de la géométrie pour ne traiter que la moitié droite du profil. A gauche, nous avons donc un axe de symétrie et le profil est posé sur un sol horizontal. Sous l'effet de la gravité, le haut du profil se déforme et la figure représente une déformation augmentée d'un facteur 10^5 : en réalité, la déformation est vraiment minime !

Il s'agira juste de construire le système linéaire global discret : la géométrie, les conditions aux limites ont déjà été définies par nos bons soins.

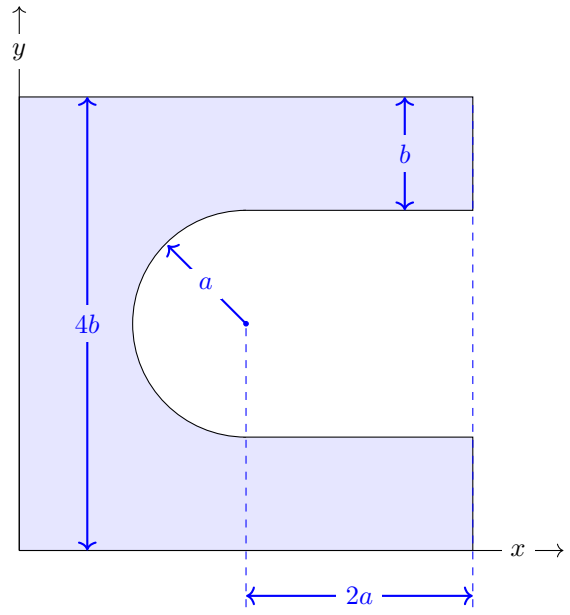
Sur le côté gauche, le déplacement horizontal est imposé à une valeur nulle.
Sur la côté inférieur, le déplacement vertical est imposé à une valeur nulle : le bloc peut glisser sur le sol.
La couleur est proportionnelle à l'amplitude ou la norme de la déformation.

Définir la géométrie et le maillage !

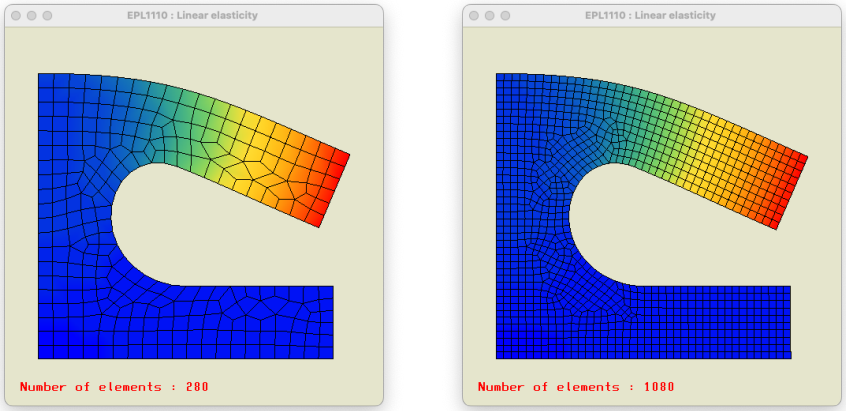
Le maillage et la géométrie sont définis comme suit :

```
double Lx = 1.0;
double Ly = 1.0;

theGeometry->LxPlate     = Lx;
theGeometry->LyPlate     = Ly;
theGeometry->h           = Lx * 0.75;
theGeometry->elementType = FEM_QUAD;
```



La densité d'éléments est fixée par la taille caractéristique `h` et le type d'élément choisi (triangles ou quads) est fixé par `elementType`. En augmentant le nombre d'éléments, on observe bien que le mailleur arrive plus aisément à obtenir des quadrilatères de meilleure qualité...



Equations de l'élasticité linéaire plane

La *formulation faible* des équations de l'élasticité linéaire s'écrit sous la forme suivante :

$$\boxed{\begin{array}{l}\text{Trouver } \mathbf{u}(\mathbf{x}) \in \mathcal{U} \text{ tel que} \\ \underbrace{< \boldsymbol{\epsilon}(\widehat{\mathbf{u}}) : \mathbf{C} : \boldsymbol{\epsilon}(\mathbf{u}) >}_{a(\widehat{\mathbf{u}},\mathbf{u})} = \underbrace{< \widehat{\mathbf{u}} f > + \ll \widehat{\mathbf{u}} g \gg_N}_{b(\widehat{\mathbf{u}})}, \qquad \forall \widehat{\mathbf{u}} \in \widehat{\mathcal{U}},\end{array}} \tag{1}$$

La fonctionnelle associée au problème de minimum équivalent est donné par

$$\boxed{J(\mathbf{v}) = \underbrace{\frac{1}{2} < \boldsymbol{\epsilon}(\mathbf{v}) : \mathbf{C} : \boldsymbol{\epsilon}(\mathbf{v}) >}_{\frac{1}{2}a(\mathbf{v},\mathbf{v})} - \underbrace{< \mathbf{v} f > + \ll \mathbf{v} g \gg_N}_{b(\mathbf{v})}} \tag{2}$$

où le premier terme correspond à l'énergie de déformation et le second au travail des forces extérieures. Cette fonctionnelle représente l'énergie que la matériau va minimiser en se déformant !

Pour les problèmes à deux dimensions, on distingue le cas des déformations planes et des tensions planes. En effet, on observe, qu'en vertu de l'équation de comportement, il est impossible de supposer simultanément que les contraintes et les déformations soient planes.

En supposant des déformations planes ($\epsilon_{zz} = \epsilon_{xz} = \epsilon_{yz} = 0$), les équations de comportement s'écrivent :

$$
\begin{aligned}
\sigma_{xx} &= \frac{E}{(1+\nu)(1-2\nu)}\left((1-\nu)\epsilon_{xx}+\nu\epsilon_{yy}\right), \\
\sigma_{yy} &= \frac{E}{(1+\nu)(1-2\nu)}\left(\nu\epsilon_{xx}+(1-\nu)\epsilon_{yy}\right), \\
\sigma_{zz} &= \frac{E\nu}{(1+\nu)(1-2\nu)}\left(\epsilon_{xx}+\epsilon_{yy}\right), \\
\sigma_{xy} &= \frac{E}{(1+\nu)}\epsilon_{xy},
\end{aligned}
\tag{3}
$$

tandis qu'en tensions planes ($\sigma_{zz} = \sigma_{xz} = \sigma_{yz} = 0$), on a

$$
\begin{aligned}
\sigma_{xx} &= \frac{E}{(1-\nu^2)}\left(\epsilon_{xx}+\nu\epsilon_{yy}\right), \\
\sigma_{yy} &= \frac{E}{(1-\nu^2)}\left(\nu\epsilon_{xx}+\epsilon_{yy}\right), \\
\sigma_{xy} &= \frac{E}{(1+\nu)}\epsilon_{xy}.
\end{aligned}
\tag{4}
$$

où on utilise le fait que la contrainte σ_{zz} est supposée nulle, pour exprimer ϵ_{zz} en termes de ϵ_{xx} et ϵ_{yy}.

Les deux systèmes (3) (à l'exception de la relation pour la contrainte transversale σ_{zz}) et (4) peuvent s'écrire sous une forme matricielle unique

$$
\begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{bmatrix} = \begin{bmatrix} A\epsilon_{xx}+B\epsilon_{yy} & 2C\epsilon_{xy} \\ 2C\epsilon_{xy} & A\epsilon_{yy}+B\epsilon_{xx} \end{bmatrix}
\tag{5}
$$

où les constantes A, B, C sont définies dans le tableau suivant.

Déformations planes	Tensions planes
$A = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}$	$A = \frac{E}{(1-\nu^2)}$
$B = \frac{E\nu}{(1+\nu)(1-2\nu)}$	$B = \frac{E\nu}{(1-\nu^2)}$
$C = \frac{E}{2(1+\nu)}$	

Nous utiliserons désormais cette formulation unique, tout en sachant qu'elle convient à la fois pour les tensions planes et les déformations planes. Avec cette formulation, l'énergie de déformation s'écrit sous la forme

$$
\begin{aligned}
\tfrac{1}{2}\, a(\mathbf{u},\mathbf{u}) &= \tfrac{1}{2} < \boldsymbol{\epsilon}(\mathbf{u}) : \boldsymbol{\sigma}(\mathbf{u}) >, \\
&= \tfrac{1}{2} < \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} \\ \epsilon_{xy} & \epsilon_{yy} \end{bmatrix} : \begin{bmatrix} A\epsilon_{xx} + B\epsilon_{yy} & 2C\epsilon_{xy} \\ 2C\epsilon_{xy} & A\epsilon_{yy} + B\epsilon_{xx} \end{bmatrix} >, \\
&= \tfrac{1}{2} < \begin{bmatrix} \epsilon_{xx} & \epsilon_{yy} & 2\epsilon_{xy} \end{bmatrix} \cdot \begin{bmatrix} A & B & 0 \\ B & A & 0 \\ 0 & 0 & C \end{bmatrix} \cdot \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ 2\epsilon_{xy} \end{bmatrix} > .
\end{aligned}
\tag{6}
$$

Cette dernière écriture n'est possible que si la composante de cisaillement ϵ_{xy} est multipliée par 2 lors de la définition du pseudo-vecteur des déformations infinitésimales remplacant le tenseur d'ordre deux. L'intérêt de cette dernière écriture est d'identifier une matrice centrale de dimension 3 qui doit être définie positive pour que la forme a le soit également.

Ensuite, il suffit de la la matrice locale de raideur dont chaque composante est une matrice 2×2.

$$
\begin{aligned}
\mathbf{A}_{ij} &= \left[\begin{array}{c|c} < \tau_{i,x} A \tau_{j,x} > + < \tau_{i,y} C \tau_{j,y} > & < \tau_{i,x} B \tau_{j,y} > + < \tau_{i,y} C \tau_{j,x} > \\ \hline < \tau_{i,y} B \tau_{j,x} > + < \tau_{i,x} C \tau_{j,y} > & < \tau_{i,y} A \tau_{j,y} > + < \tau_{i,x} C \tau_{j,x} > \end{array}\right], \\
\mathbf{B}_i &= \begin{bmatrix} < \tau_i f_x > + \ll \tau_i g_x \gg \\ < \tau_i f_y > + \ll \tau_i g_y \gg \end{bmatrix}.
\end{aligned}
$$

Pour effectuer une simulation, il ne reste plus qu'à trouver les paramètres matériels de l'acier :

Acier
$E = 2.11\ 10^{11}\ [N/m^2]$
$\nu = 0.3$
$\rho = 7.85\ 10^3\ [kg/m^3]$

Implémentation de l'élasticité linéaire plane

Les données du problèmes sont stockées dans la structure suivante :

```
typedef struct {
    femDomain* domain;
    femBoundaryType type;
    double value;
} femBoundaryCondition;

typedef struct {
    double E,nu,rho,g;
    double A,B,C;
    int planarStrainStress;
    int nBoundaryConditions;
    femBoundaryCondition **conditions;
    int *constrainedNodes;
    femGeo *geometry;
    femDiscrete *space;
    femIntegration *rule;
    femFullSystem *system;
} femProblem;
```

Ensuite, quelques fonctions permettent de résoudre notre problème d'élasticité linéaire.

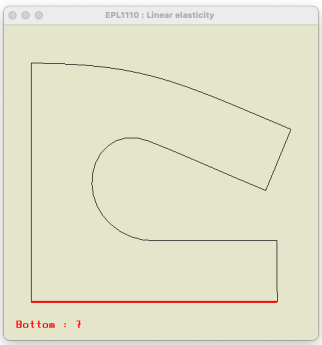
```
femProblem*         femElasticityCreate(femGeo* theGeometry,
                              double E, double nu, double rho, double g, femElasticCase iCase);
void                femElasticityFree(femProblem *theProblem);
void                femElasticityPrint(femProblem *theProblem);
void                femElasticityAddBoundaryCondition(femProblem *theProblem,
                              char *nameDomain, femBoundaryType type, double value);
double*             femElasticitySolve(femProblem *theProblem);
```

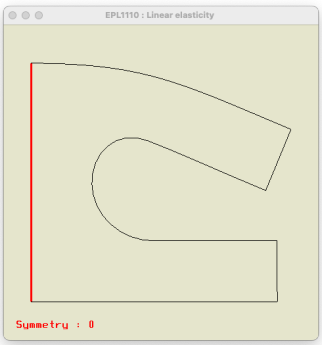
1. La fonction `femElasticityCreate` crée un problème à partir d'un maillage avec les paramètres matériels requis. En fonction du type de maillage (triangles ou quads), on sélectionne des fonctions de forme linéaires de Turner ou des fonctions de forme bilinéaires. On procède de même pour le choix de la règle d'intégration. Avec les fonctions `femElasticityFree` et `femElasticityPrint`, on libére la mémoire allouée et on imprime les données définies : ce qui est parfois bien utile.

2. La fonction `femElasticityAddBoundaryCondition` permet de définir une condition frontière essentielle sur un domaine identifé par un nom !

3. La fonction `femElasticitySolve` permet d'obtenir la solution.
   Malencontreusement, un assistant facétieux a subtilisé une partie du code :-(

Et concrètement ?





Concrètement, la définition et la résolution du problème est obtenue de la manière suivante :

```
geoInitialize();
femGeo* theGeometry = geoGetGeometry();
theGeometry->LxPlate     =  Lx;
theGeometry->LyPlate     =  Ly;
theGeometry->h           =  Lx * 0.05;
theGeometry->elementType = FEM_QUAD;
geoMeshGenerate();
geoMeshImport();
geoSetDomainName(0,"Symmetry");
geoSetDomainName(7,"Bottom");

double E   = 211.e9;
double nu  = 0.3;
double rho = 7.85e3;
double g   = 9.81;
femProblem* theProblem = femElasticityCreate(theGeometry,E,nu,rho,g,PLANAR_STRAIN);
femElasticityAddBoundaryCondition(theProblem,"Symmetry",DIRICHLET_X,0.0);
femElasticityAddBoundaryCondition(theProblem,"Bottom",DIRICHLET_Y,0.0);
femElasticityPrint(theProblem);
double *theSoluce = femElasticitySolve(theProblem);
```

1. Il faut donc simplement implémenter la fonction suivante :

   ```
   double *femElasticitySolve(femProblem *theProblem));
   ```

   Plus précisément, il faut construire le système linéaire discret pour le problème défini. L'imposition des conditions frontières essentielles a été réalisé par nos soins : il ne faut donc rien réaliser :-)

2. Vos deux fonctions seront incluses dans un unique fichier `homework.c`, sans y adjoindre le programme de test fourni ! Ce fichier devra être soumis via le web et la correction sera effectuée automatiquement. Il est donc indispensable de respecter strictement la signature des fonctions. Votre code devra être strictement conforme au langage `C` et il est fortement conseillé de bien vérifier que la compilation s'exécute correctement sur le serveur.