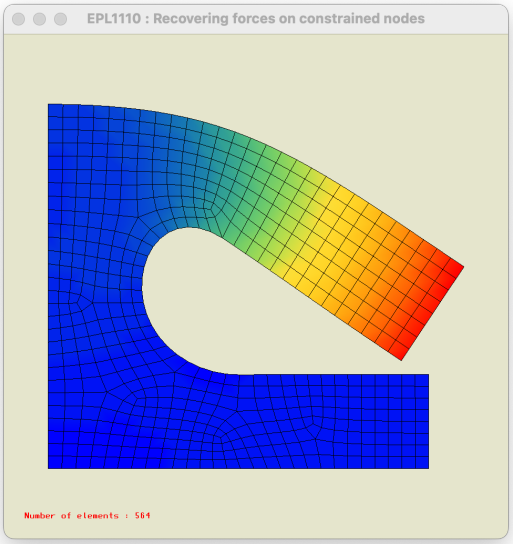
Finite elements for dummies : Elasticité linéaire !



Ce dernier devoir est consacré à l'implémentation des conditions de Neumann et au calcul de la densité des forces de contact lorsque des conditions de Dirichlet sont appliquées.

Nous allons toujours 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 !

La situation est toutefois très légèrement modifiée car on y ajoute une densité de charge constante de 10^4 $[N/m^2]$.

Sur le côté gauche, le déplacement horizontal est imposé à une valeur nulle.
Sur le coté inférieur, une densité de charge est appliquée.
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.

Calcul des intégrales de surface

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

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

La matrice de raideur et le membre de droite peuvent être vues comme des hyper-matrice et vecteur dont dont chaque composante est une matrice 2×2 ou un vecteur de taille de dimension 2.

$$\mathbf{A}_{ij} = \left[\begin{array}{c|c} < \tau_{i,x} A \tau_{j,x} > + < \tau_{i,y} C \tau_{j,y} > & < \tau_{i,x} B \tau_{j,y} > + < \tau_{i,y} C \tau_{j,x} > \\ \hline < \tau_{i,y} B \tau_{j,x} > + < \tau_{i,x} C \tau_{j,y} > & < \tau_{i,y} A \tau_{j,y} > + < \tau_{i,x} C \tau_{j,x} > \end{array}\right],$$

$$\mathbf{B}_i = \left[\begin{array}{c} < \tau_i f_x > + \ll \tau_i g_x \gg \\ \\ < \tau_i f_y > + \ll \tau_i g_Y \gg \end{array}\right].$$

Dans ce devoir, il s'agira de calculer les intégrales de ligne lorsqu'on a une condition de Neumann. Il s'agira ensuite d'estimer les forces de contact qu'il faudrait appliquer si on remplaçait une condition essentielle par une condition naturelle équivalente. Pour obtenir cela, on procédera de la manière suivante : on estimera les résidus des équations non-contraintes du système linéraire pour la solution obtenue. Pour tous les noeuds contraints, les équations correspondantes ont été remplacées par les contraintes et ne sont donc pas satisfaites par la solution. Elles ne sont pas satisfaites car on n'y a pas calculé les intégrales de ligne.... et donc c'est bien logique. Mais, ce terme qu'on a omis et qui donne lieu à l'apparition du résidu est justement celui que l'on souhaite calculer : il suffira donc de prendre l'opposé du résidu pour avoir la densité des forces nodales exercées sur chaque noeud contraint. Pour obtenir la force de contact globale, il suffira alors d'accumuler toutes les forces nodales !

Quelques légères adaptations de la structure de données

La structure de données a été légèrement enrichie pour pouvoir réaliser notre devoir. En particulier, on y a ajouté deux vecteurs contenant la solution trouvée et le résidu. On y a aussi ajouté un espace discret unidimensionnel et une règle d'intégration unidimensionnelle pour calculer les intégrales de ligne.

```
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;
    double *soluce;
    double *residuals;
    femGeo *geometry;
    femDiscrete *space;
    femIntegration *rule;
    femDiscrete *spaceEdge;
    femIntegration *ruleEdge;
    femFullSystem *system;
} femProblem;
```

Ensuite, nous avons aussi légèrement modifié les fonctions de notre problème

```
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);
void                femElasticityAssembleElements(femProblem *theProblem);
void                femElasticityAssembleNeumann(femProblem *theProblem);
double*             femElasticitySolve(femProblem *theProblem);
double*             femElasticityForces(femProblem *theProblem);
double              femElasticityIntegrate(femProblem *theProblem, double (*f));
```

1. La fonction `femElasticityAddBoundaryCondition` permet de définir une condition frontière. Il peut s'agir de condition essentielle ou naturelle sur un domaine identifié par un nom !

2. La fonction `femElasticityAssembleElements` calcule et assemble toutes les intégrales de surface. Ce sont toutes les intégrales de la formulation discrète à l'exception de celles de Neumann.

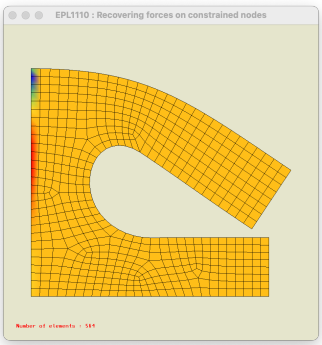
3. La fonction `femElasticityAssembleNeumann` calcule et assemble toutes les intégrales de ligne. Ce sont les intégrales associées aux conditions de Neumann de la formulation discrète.

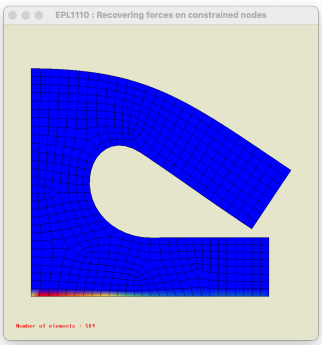
4. Le fonction `femElasticitySolve` permet d'obtenir la solution.
   Cette solution est stockée dans le vecteur `soluce`.
   La fonction renvoie le pointeur de ce vecteur.

5. Le fonction `femElasticityForces` permet d'obtenir le résidus des équations non contraintes.
   Elle renvoie le pointeur de ce vecteur.

6. La fonction `femElasticityIntegrate` permet d'intégrer la surface.
   Cela doit permettre d'obtenir le poids de la poutre.

Et concrètement ?





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

```
    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);
    femElasticityAddBoundaryCondition(theProblem,"Top",NEUMANN_Y,-1e4);
    femElasticityPrint(theProblem);

//
//  -3- Resolution du probleme et calcul des forces
//

    double *theSoluce = femElasticitySolve(theProblem);
    double *theForces = femElasticityForces(theProblem);
    double area = femElasticityIntegrate(theProblem, fun);

//
//  -4- Deformation du maillage pour le plot final
//      Creation du champ de la norme du deplacement
//

    femNodes *theNodes = theGeometry->theNodes;
    double deformationFactor = 1e5;
    double *normDisplacement = malloc(theNodes->nNodes * sizeof(double));
    double *forcesX = malloc(theNodes->nNodes * sizeof(double));
    double *forcesY = malloc(theNodes->nNodes * sizeof(double));

    for (int i=0; i<theNodes->nNodes; i++){
        theNodes->X[i] += theSoluce[2*i+0]*deformationFactor;
        theNodes->Y[i] += theSoluce[2*i+1]*deformationFactor;
        normDisplacement[i] = sqrt(theSoluce[2*i+0]*theSoluce[2*i+0] +
                                   theSoluce[2*i+1]*theSoluce[2*i+1]);
        forcesX[i] = theForces[2*i+0];
        forcesY[i] = theForces[2*i+1]; }
```

```
    double hMin = femMin(normDisplacement,theNodes->nNodes);
    double hMax = femMax(normDisplacement,theNodes->nNodes);
    printf(" ==== Minimum displacement          : %14.7e [m] \n",hMin);
    printf(" ==== Maximum displacement          : %14.7e [m] \n",hMax);

//
//  -5- Calcul de la force globale resultante
//

    double theGlobalForce[2] = {0, 0};
    for (int i=0; i<theProblem->geometry->theNodes->nNodes; i++) {
        theGlobalForce[0] += theForces[2*i+0];
        theGlobalForce[1] += theForces[2*i+1]; }
    printf(" ==== Global horizontal force       : %14.7e [N] \n",theGlobalForce[0]);
    printf(" ==== Global vertical force         : %14.7e [N] \n",theGlobalForce[1]);
    printf(" ==== Weight                        : %14.7e [N] \n", area * rho * g);
```

Observer qu'il semble nécessaire de recalculer la matrice de raideur globale pour calculer le résidu ! Ce n'est pas la meilleure idée, il est sans doute été judicieux de conserver la matrice construite avant de résoudre le système linéaire en place, au lieu de la recalculer. Avec un peu d'astuce, vous pouvez éviter de faire ce calcul en copiant la matrice dans une structure intermédiaire que vous pouvez définir vous-mêmes. Evidemment, c'est un peu compliqué, mais cela devrait rendre votre code plus rapide...

1. Tout d'abord, il faut implémenter la fonction suivante :

   ```
   void femElasticityAssembleNeumann(femProblem *theProblem);
   ```

   Plus précisément, il faut calculer les intégrales de ligne associées aux conditions de Neumann. Et ensuite les assembler.

2. Ensuite, il faut implémenter la résolution du problème d'élasticité

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

   En gros, c'est exactement la même fonction que celle du devoir précédent, mais c'est une bonne idée de faire ici appel à `femElasticityAssembleElements` et `femElasticityAssembleNeumann`

3. Finalement, il s'agit de calculer les résidus de votre problème.

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

   On peut ici à nouveau faire appel aux deux fonctions d'assemblage ou récupérer astucieusement la matrice de raideur. On supppose évidemment qu'on dispose de la solution `soluce` dans la structure du problème et donc qu'on a fait appel à la fonction précédente au préalable.

4. Vos trois 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.