LINMA1170 – Devoir 2 : Solveur Bande

Vous êtes maintenant équipés d'une structure de matrice pleine ainsi que d'un solveur de type LU plein pour résoudre des systèmes linéaires. Dans ce devoir, on vous propose d'appliquer vos algorithmes à un problème classique en mécanique du solide : **la déformation en élasticité linéaire**.

Afin d'accélérer le calcul, vous allez tirer profit de l'aspect creux de la matrice et implémenter un solveur bande. Grâce à une numérotation des nœuds judicieuse, vous obtiendrez une complexité théorique bien meilleure que le solveur plein.

Problème de déformation

Vous allez étudier le problème de la déformation d'un carré soumis à une force sur un de ses côtés. En élasticité linéaire, la déformation est directement proportionnelle à la force appliquée. La géométrie du carré est discrétisée à l'aide d'un maillage, et les inconnues du problèmes sont les déformations $\mathbf{U}^i = (U_x^i, U_y^i)$ en chaque nœud i du maillage ; pour un maillage à n nœuds il y a donc $2n$ inconnues. La déformation obéit au système linéaire

$$KU = F,$$

où $K \in \mathbb{R}^{2n \times 2n}$ est la *matrice de raideur*, et $F \in \mathbb{R}^{2n}$ est le *vecteur des forces nodales* qui correspond aux forces appliquées sur l'objet.

Vous ne devez pas vous-mêmes calculer les matrices d'éléments finis K et F : c'est l'objet du cours homonyme. On vous fournit donc un module `elasticity` qui fait le calcul pour vous. Prenez bien connaissance du header correspondant afin de comprendre le format des données fournies.

Phénomène de *fill-in*

Une matrice *creuse* est une matrice possédant beaucoup plus d'éléments nuls que d'éléments non nuls. On pourrait s'attendre à ce que calculer la décomposition LU d'une matrice creuse soit rapide, mais ce n'est pas vrai en général. Pour s'en rendre compte, il faut étudier le *masque* (la carte des éléments non nuls) : même si la matrice a un masque creux, le masque de sa décomposition LU ne l'est pas. C'est ce qu'on appelle le *fill-in*, ou *remplissage*. On vous demande dans un premier temps de mettre en évidence ce phénomène sur le problème donné.

Numérotation des inconnues

Une stratégie bien connue pour atténuer le *fill-in* est de permuter les inconnues afin de réduire le remplissage. L'idéal est que les éléments non nuls se retrouvent le plus proche possible de la diagonale de la matrice. Ainsi le remplissage se fait également plus proche de la diagonale, et on peut tirer profit d'une structure *bande* de la matrice.

Stockage bande

Soit une matrice $A \in \mathbb{R}^{m \times m}$. On définit sa bande comme la zone autour de la diagonale principale contenant toutes les entrées non-nulles de la matrice. Autrement dit, la largeur de bande est k si

$$a_{ij} = 0 \quad \text{pour} \quad |i - j| > k.$$

Une matrice est dite *bande* si k est petit devant m.

Dans une matrice bande, chacune des m lignes contient au maximum $2k + 1$ entrées. Plusieurs formats existent pour stocker des matrices creuses. Le format le plus simple consiste à stocker l'entièreté de la bande, c'est-à-dire les $2k + 1$ entrées. Vous allez donc créer une nouvelle structure de données :

```
typedef struct BandMatrix {
    int m, k;       // dimension de la matrice et largeur de bande
    double * data;  // tableau 1D de taille m*(2*k+1) contenant les entrées de la matrice
    double ** a;    // tableau 1D de m pointeurs vers chaque ligne, pour pouvoir appeler a[
} BandMatrix;
```

Attention : on veut garder la propriété que `a[i][j]` corresponde à a_{ij}. Pour cela, il faudra définir judicieusement le pointeur `a[i]` !

Ce qui vous est demandé

1. On vous donne un script `solve_deformation.c` qui ouvre une géométrie, crée un maillage, calcule le système linéaire, le résout et affiche la solution. Couplez ce code avec votre implémentation du devoir 1 et visualisez la solution.
2. Mettez en évidence le phénomène de *fill-in*. Une manière de procéder est d'écrire la matrice dans un fichier CSV, puis d'utiliser Python pour plotter son masque (avec `numpy.loadtxt` et `matplotlib.pyplot.spy`, par exemple). Quel est le taux de remplissage de la matrice avant et après décomposition LU ?
3. Implémentez une stratégie de renumérotation dans `compute_permutation`. On vous propose comme stratégie de trier les nœuds par leur coordonnées en `x`, mais vous pouvez implémenter des stratégies plus compliquées en guise de bonus.
4. Montrez le *fill-in* avec la nouvelle numérotation et commentez. Quelle est la largeur de bande de la matrice ? Commentez par rapport à la statégie de numérotation choisie.
5. Implémentez les fonctions `allocate_band_matrix`, `free_band_matrix`, `lu_band` et `solve_band` pour pouvoir travailler avec des matrices bandes.
6. Analysez le gain spatial et temporel du stockage bande. Quelle complexité obtenez-vous pour le calcul de la LU et la résolution du système ? Commentez.