Résumé

Ce rapport traite et discute des avantages des solveurs bandes dans la résolution de grands systèmes linéaires. Nous illustrerons ces gains théoriques par un exemple simple de déformation en élasticité linéaire, i.e. la déformation d'un carré soumis à une force sur l'un de ses côtés. Afin de résoudre ce problème mécanique et de mener les expérimentations, plusieurs scripts ont été rédigés. Pour toute information subsidiaire quant à l'utilisation de ces derniers, le lecteur est invité à consulter le `README` associé au projet.

Phénomène de *fill-in* et permutations

La figure 1 montre le masque de la *matrice de raideur* avant et après la décomposition LU selon un maillage de 228 noeuds généré. Le taux de remplissage initial vaut 0.03%. Après la factorisation, il est mesuré à 0.27%. Le phénomène de remplissage est donc considérable.Ces résultats sont indépendant du nombre de noeuds choisis. En

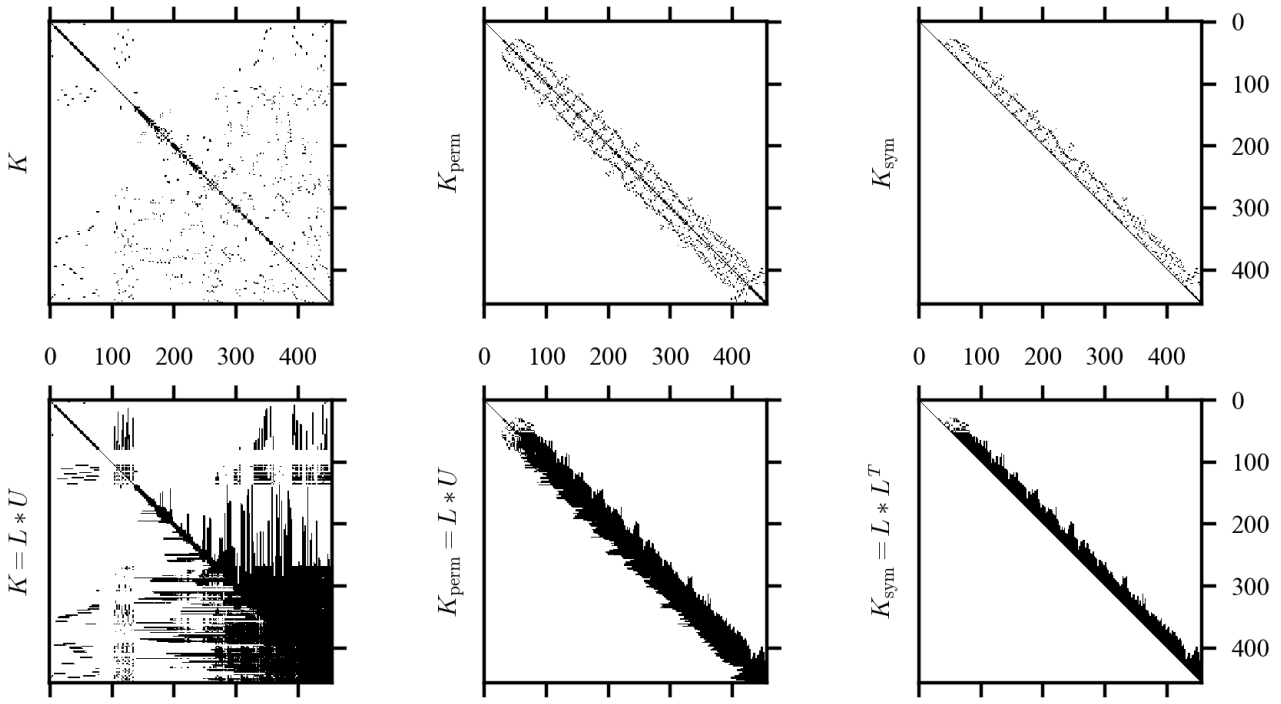

FIGURE 1 – Masques des *matrices de raideur* avant et après la décomposition LU.

comparant les masques, nous pouvons observer que les valeurs de K loin de la diagonale produisent des bandes de valeurs qui les relient à la diagonale. Une renumérotion astucieusement des noeuds est donc nécessaire afin de rappprocher l'ensemble des coordonnées des noeuds de la diagonale. Pour ce faire, la stratégie de renumérotation fondée sur le tri des noeuds selon leur coordonnée en `x` puis selon leur coordonnée en `y` en cas d'égalité a été implémentée dans `compute_permutation`. En appliquant les permutations sur K, on obtient une nouvelle matrice K_{perm}. La figure 1 montre que cette stratégie est efficace et permet de réduire considérablement la largeur de bande de la *matrice de raideur*. Ceci n'est guère surprenant au vu de la géométrie du problème mécanique. En effet, la force étant appliqué sur le côté droit du carré, les noeuds ayant une coordonnées en `x` proches sont enclins à interagir entre eux.

Le taux de remplissage après la factorisation de K_{perm} est de 0.12%. Outre la réduction notable, l'ensemble des valeurs de la factorisation reste cantonné dans la bande de la matrice. Il est ainsi possible de ne stocker que cette

dernière au lieu de la matrice entière. En tirant profit du caractère symétrique de la matrice, il est en réalité possible de ne stocker que la moitié de la bande (K_{sym}). Les possibilités qu'offre cette symétrie sont discutées plus en détails dans la section suivante.

Perfomances

Commençons par déduire la complexité de l'algorithme `lu_band`. Soit β la largeur de bande d'une matrice à décomposer $A^{m\times m}$ tel que $a_{ij} = 0$ pour $|i-j| > \beta$. Lors de la k-ième itération de la décomposition, nous mettons à jour une sous-matrice de dimension $(\beta+1)\times(\beta+1)$ dont le premier élément correspond au k-ième élément diagonal de A. Ceci requiert donc $2(\beta+1)^2$ opérations en virgule flottante. Toutefois, cette sous-matrice doit pouvoir être entièrement contenue dans A. La sous-matrice sera donc de dimension $(m-k-1)\times(m-k-1)$ pour les β dernières itérations. Formellement, le nombre d'opérations total vaut

$$\sum_{k=0}^{m-\beta-1} 2(\beta+1)^2 + \sum_{k=m-\beta}^{m-1} 2(m-k-1)^2 = 2(\beta+1)^2(m-\beta) + \frac{1}{3}\beta(2\beta^2-3\beta+1) \sim 2m\beta^2$$

Ainsi, notre terme dominant dépend évidemment de la largeur de bande de la matrice. Toutefois, nous aimerions exprimer β en fonction de m afin de pouvoir comparer théoriquement notre décomposition bande avec la décomposition LU classique. Expérimentalement, les mesures suivantes ont été relevées sur la *matrice de raideur* $K^{m\times m}$ en faisant varier le nombre de noeuds composant le maillage.

m	10	24	60	288	1022	1578	2880	3868
β	5	11	19	45	85	103	141	163

En représentant ces paires de valeurs sur un repère orthonormé, on observe assez facilement que $\beta \approx 2.6\sqrt{m}$. Dès lors, nous pouvons réécrire le terme dominant **pour ce problème mécanique** comme $\sim 13.52m^2$. Comme vu au premier devoir, la complexité temporelle de l'algorithme `lu` classique est $\sim \frac{2}{3}m^3$. Nous l'avons donc réduite d'un ordre de grandeur [1] ! La figure 2 confirme nos suppositions car les comportements asymptotiques concordent.

En ce qui concerne la fonction `solve_band`, nous pouvons calculer le nombre d'opérations en virgule flottante de manière analogue. Pour un système linaire inférieurement [2] triangulaire, lors des β premières itérations, la résolution de la k-ième équation requiert 1 division, k soustractions et k multiplications. Pour les itérations suivantes, il ne suffit plus que de 1 division, β soustractions et β multiplications. Dès lors, le nombre d'opérations total est donné par

$$2\sum_{k=0}^{\beta-1} 2k+1 + 2\sum_{k=\beta}^{m-1} 2*\beta+1 = 2\beta^2 + 2(2\beta+1)(m-\beta) \sim 4m\beta$$

Or nous savons [3] que `solve` $\sim 2m^2$. En utilisant la relation déduite au paragraphe précédent, il vient que `solve_band` $\sim 10.4m^{\frac{3}{2}}$ pour ce problème. Les bénéfices temporelles sont donc moins importants puisque que nous n'avons gagné qu'un demi ordre de grandeur. A nouveau, les mesures confirment les hypothèses [4] (Fig. 2).

Enfin, la structure `BandMatrix` offre un gain spatial considérable puisqu'elle permet de ne stocker au plus seulement $m(2\beta+1)$ valeurs soit $8m(2\beta+1) \approx 8m(5.2\sqrt{m}+1)$ bytes contre les $8m^2$ initiaux. Par ailleurs, la

1. Le calcul des permutations ($\sim \frac{m}{2}\log\frac{m}{2}$) n'a pas été inclus au calcul car jugé négligeable.
2. Pour un système supérieurement triangulaire, on obtient le même résultat en commençant par la dernière équation.
3. Voir devoir 1.
4. Comme détaillé au premier devoir, le décalage entre la courbe théorique et expérimentale s'explique par de nombreux facteurs : latence des accès mémoire, coeurs du processeur partagés, fil d'instructions non-parallélisable, ...

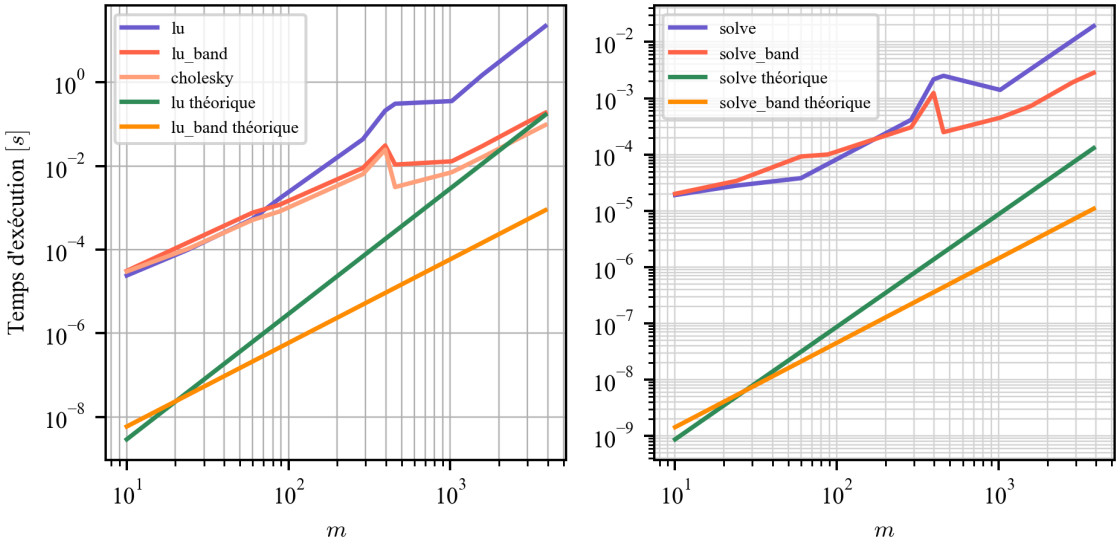

FIGURE 2 – Mesures des performances des algorithmes.

matrice de raideur est définie-positive et symétrique par construction. D'une part, il est donc possible de ne stocker que $m(\beta+1)$ valeurs[5] en adaptant légèrement nos fonctions `lu_band` et `solve_band`[6]. D'autre part, nous pouvons utiliser l'algorithme de Cholesky (voir `cholesky`) afin de calculer la décomposition $K_{\text{sym}} = L * L^T$ deux fois plus rapidement (Fig. 2). La fonction `solve_cholseky` résout le système linéaire associé mais n'apporte aucun avantage temporel par rapport à `solve_band`.

Fonctions

La liste ci-dessous explicite brièvement les fonctions et structures de données les plus importantes du projet.

- `Matrix` : ADT représentant une matrice $\mathbb{R}^{m\times n}$.
- `BandMatrix` : ADT représentant une matrice $\mathbb{R}^{m\times m}$ de largeur de bande k.
- `SymBandMatrix` : ADT représentant une matrice symétrique $\mathbb{R}^{m\times n}$ de largeur de bande k.
- `lu(Matrix *A)` : Calcule $A = L * U$.
- `solve(Matrix *LU, double *y)` : Résout $L * U * x = y$.
- `lu_band(BandMatrix *A)` : Calcule $A = L * U$.
- `solve_band(BandMatrix *LU, double *y)` : Résout $L * U * x = y$.
- `cholesky(SymBandMatrix *A)` : Calcule $A = L * L^T$.
- `solve_cholesky(SymBandMatrix *LLT, double *y)` : Résout $L * L^T * x = y$.

Pour s'assurer de la correction de ces fonctions, les vecteurs solutions obtenus ont été comparés avec celui produit par les fonctions du premier devoir dont l'exactitude a été attestée.
D'autres fonctions auxiliaires sont disponibles dont la plupart permettent d'afficher les ADTs sur la sortie standard ou de les sauvegarder dans des fichiers `.csv`. Un `Makefile` est aussi fourni dont les cibles disponibles sont décrites dans le `README`. La principale, `make`, résout le problème d'elasticité linéaire en employant séquentiellement les différentes ADTs ci-dessus et leurs fonctions associées puis affiche la solution.

5. Une structure additionnelle `SymBandMatrix` exploitant ceci a été implémentée dans `matrix.h`.
6. Si nous ne stockons que la partie supérieure de la bande, a_{ij} = `(i <= j) ? A[i][j] : A[j][i]`.