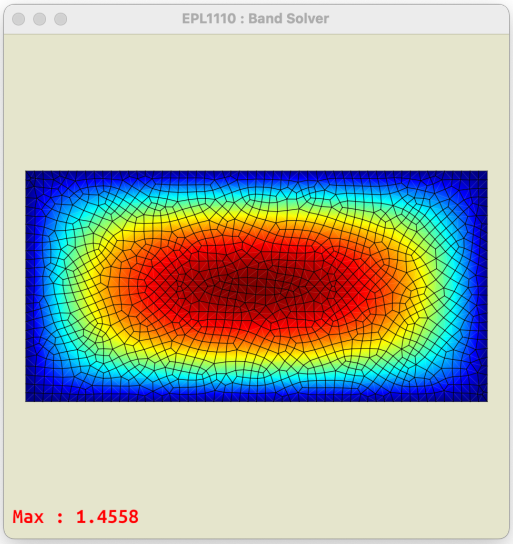
Finite elements for dummies : Résolution d'un système linéaire creux...

Nous allons reprendre notre premier programme d'éléments finis et tenter de le rendre un peu plus efficace ! Nous utilisons toujours des éléments triangulaires linéaires continus ou des éléments quadrilatères bilinéaires continus pour résoudre le problème suivant :

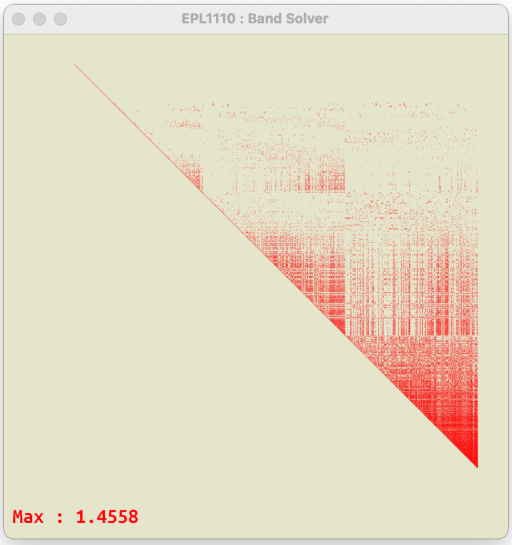


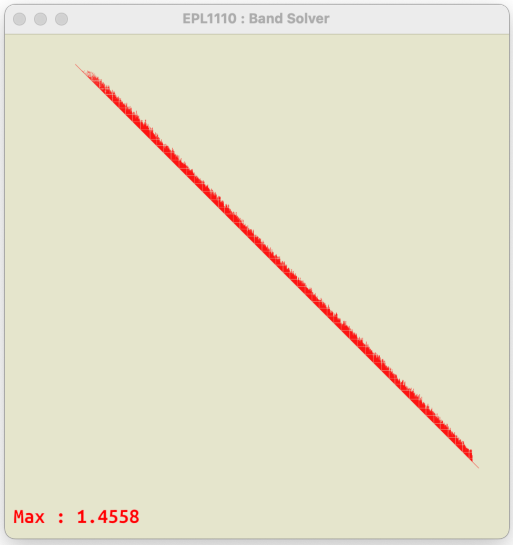
Trouver $u(x,y)$ tel que

$$\begin{aligned} \nabla^2 u(x,y) + 1 &= 0, & \forall (x,y) \in \Omega, \\ u(x,y) &= 0, & \forall (x,y) \in \partial\Omega, \end{aligned}$$

Il s'agit maintenant de comparer les performances de divers solveurs pour la résolution du système linéaire discret. Nous allons comparer les performances d'un solveur direct plein et d'un solveur direct bande. On vous fournit aussi un solveur itératif assez élémentaire : le prochain exercice sera consacré à l'implémentation des gradients conjugés : ne soyez pas impatients :-)

En utilisant le clavier comme sur une console de jeu élémentaire, vous pouvez, de manière interactive, changer le solveur utilisé et l'algorithme de renumérotation sans devoir recompiler le code. Il sera toujours possible de voir la solution ou d'inspecter l'allure de la matrice. L'exercice consiste à programmer, mais aussi à manipuler votre programme pour comprendre tout l'intérêt d'un solveur bande avec une bonne numérotation de noeuds. N'hésitez donc pas à utiliser le programme avec les maillages `triangles101.txt`, `triangles101xopt.txt` et `triangless101yopt.txt`. où trois numérotations distinctes ont été utilisées dans la numérotation de noeuds. En comparant les résultats obtenus avec le solveur bande, on observe que bien renuméroter les noeuds est critique.





Mais pour les autres maillages, ce sera votre travail de renuméroter les noeuds.

- Pour la résolution du système linéaire, vous disposerez de trois solveurs avec une interface commune. Il est donc possible d'avoir une implémentation unique du problème, en pouvant faire appel à n'importe lequel des trois solveurs et pouvant modifier ce choix sans devoir recompiler le programme. Toutes les fonctions permettant l'allocation, l'initialisation, l'assemblage et la résolution d'un système linéaire ont été ajoutées dans le module `fem`. Il est aussi possible d'imprimer la forme courante du système linéaire : ce qui peut être très utile pour mettre au point de votre programme en utilisant un maillage élémentaire.

```
femSolver*           femSolverFullCreate(int size);
femSolver*           femSolverBandCreate(int size,int band);
femSolver*           femSolverIterativeCreate(int size);
void                 femSolverFree(femSolver* mySolver);
void                 femSolverInit(femSolver* mySolver);
void                 femSolverPrint(femSolver* mySolver);
void                 femSolverPrintInfos(femSolver* mySolver);
double*              femSolverEliminate(femSolver* mySolver);
void                 femSolverAssemble(femSolver* mySolver, double *Aloc, double *Bloc,
                                                            double *Uloc, int *map, int nLoc);
double               femSolverGet(femSolver* mySolver, int i, int j);
int                  femSolverConverged(femSolver *mySolver);

femFullSystem*       femFullSystemCreate(int size);
femBandSystem*       femBandSystemCreate(int size, int band);
femIterativeSolver*  femIterativeSolverCreate(int size);
```

  Il s'agit d'une implémentation de l'héritage dans la syntaxe du C. On observe ainsi que la structure `femSolver` permet d'accèder de manière transparente à l'un des trois solveurs sans que le problème doive connaître a priori la structure utilisée.

Plus précisément, on vous demande de concevoir, d'écrire ou de modifier quatre fonctions.

1. Tout d'abord, il s'agit de réaliser l'assemblage et l'élimination gaussienne pour le solveur bande. Eh oui, un enseignant facétieux a subtilisé les deux fonctions et vous demande de les retrouver. Ce n'est pas très compliqué et c'est une bonne idée de s'inspirer de la version fournie pour un solveur plein. Les modifications à apporter sont marginales mais toute erreur se paiera par une explosion thermonucléaire de votre programme sous la forme d'une faute de segmenation ou d'un résultat incorrect. Il faut donc implémenter les deux fonctions suivantes :

```
void femBandSystemAssemble(femBandSystem* myBandSystem,double *Aloc,double *Bloc,int *map,int nLoc)
double *femBandSystemEliminate(femBandSystem *myBandSystem)
```

2. Ensuite, vous mettrez au point une fonction

```
int femMeshComputeBand(femMesh *theMesh)
```

   qui calcule la largeur de bande de la matrice du système discret associé à un maillage. La largeur de bande d'une matrice A_{ij} est la plus grande distance à la diagonale que peut atteindre un élément non nul de la matrice, augmentée d'une unité, ce que nous écrirons

$$\beta(\mathbf{A}) - 1 = \max_{ij} \{|i-j|, \quad \forall (i,j) \text{ tels que } a_{ij} \neq 0\}.$$

   Pour le moment, la version actuelle de la fonction renvoie simplement, la taille de la matrice. Ce n'est évidemment pas la meilleure manière de tirer profit du solveur bande. Comme l'assemblage se fait avant l'application des conditions aux frontières, il n'est pas possible de tenir compte que la largeur de bande est éventuellement réduite lors de l'application des conditions frontières : il faut donc juste appliquer la formule telle que donnée ci-dessus.

3. Finalement, il s'agira d'optimiser la numérotation pour réduire la largeur de bande.

```
void femMeshRenumber(femMesh *theMesh, femRenumType renumType)
```

Les indices de variables que l'on va associer à chaque noeud seront incluses dans le tableau `myMesh->number` qui est initialisé par défaut comme suit :

```
for (i = 0; i < theMesh->nNode; i++)
     theMesh->number[i] = i;
```

Cela correspond donc à ne pas renuméroter les noeuds !
Trois stratégies de renumérotations seront considérées en fonction de `renumType`.

- Ne rien faire (`FEM_NO`),
- Numéroter les noeuds en ordre croissant des abscisses (`FEM_XNUM`),
- Numéroter les noeuds en ordre croissant des ordonnées (`FEM_YNUM`).

Afin de pouvoir vérifier votre algorithme de renumérotation, nous avons fourni le maillage en y ayant appliqué les trois renumérotations : il s'agit `triangles101.txt`, `triangles101xopt.txt` et `triangless101yopt.txt` En n'appliquant aucune renumérotation sur un maillage déjà numéroté de manière optimale, vous obtiendrez la même largeur de bande que sur un maillage non numéroté auquel vous avez appliqué une renumérotation : c'est simple, non ?

4. Vos quatre 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.