# Séance 5

# Solveurs linéaires directs creux

Trouver $\mathbf{x} \in \mathbb{R}^n$ tel que

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

Trouver $\mathbf{x} \in \mathbb{R}^n$ tel que

$$J(\mathbf{x}) = \min_{\mathbf{v} \in \mathbb{R}^n} \underbrace{\left(\tfrac{1}{2}\mathbf{v} \cdot \mathbf{A}\mathbf{v} - \mathbf{b} \cdot \mathbf{v}\right)}_{J(\mathbf{v})}$$

**13** On considère la matrice tridiagonale $A$ de taille $n \times n$ définie par :

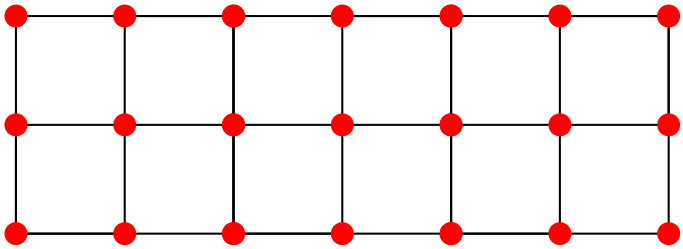
$$\mathbf{A} = \begin{bmatrix} 2 & -1 & & & & & & \\ -1 & 2 & -1 & & & & & \\ & -1 & 2 & -1 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & -1 & 2 & -1 & & \\ & & & & -1 & 2 & -1 & \\ & & & & & -1 & 2 & \end{bmatrix}$$

Comme cette matrice est symétrique et définie positive, il est possible de faire un décomposition de Cholesky $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ et la matrice $\mathbf{L}$ peut s'écrire sous la forme de deux vecteurs de taille $n$ et $n-1$ respectivement.

$$\mathbf{L} = \begin{bmatrix} d_1 & & & & & & & \\ e_1 & d_2 & & & & & & \\ & e_2 & d_3 & & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & e_{n-3} & d_{n-2} & & & \\ & & & & e_{n-2} & d_{n-1} & & \\ & & & & & e_{n-1} & d_n & \end{bmatrix}$$

1. Détailler l'algorithme de Cholesky pour cette matrice.
2. Dans cet exemple particulièrement simple, obtenir les expressions explicites de $d_j$ et $e_j$.
3. En utilisant les expressions obtenues pour $d_j$ et $e_j$, écrire une algorithm pour calculer $\mathbf{A}^{-1}\mathbf{B}$ où $\mathbf{B}$ est une matrice de taille de $n \times n$ en n'utilisant qu'un unique tableau à $n^2$ éléments à savoir les coefficients de la matrice $\mathbf{B}$. On vous fournira donc à la fonction un tableau avec les coefficients de $\mathbf{B}$ et la fonction renverra ce même tableau avec les valeurs de $\mathbf{A}^{-1}\mathbf{B}$. Il est toutefois permis d'allouer deux vecteurs en cours de calcul pour y stocker $d_j$ et $e_j$.

**14** Nous souhaitons résoudre une équation de Poisson avec des éléments bilinéaires continus sur le maillage suivant :



1. Pour un solveur linéaire creux bande ou frontal,
   est-ce la numérotation des noeuds ou des éléments qui est cruciale ?
2. Numéroter de manière optimale les élements et les noeuds de ce maillage
   pour que le coût calcul et mémoire d'un solveur bande ou frontal soit minimal ?
3. Calculer la largeur de bande du problème.
4. Calculer la largeur de front du problème.
5. Quel est la complexité d'un élimination gaussienne avec un solveur plein ?
6. Quel est la complexité d'un élimination gaussienne avec un solveur bande ?

**15** Considérons le fragment de code informatique :

```
femFullSystem *femFullSystemCreate(int size) {
    femFullSystem *theSystem = malloc(sizeof(femFullSystem));
    theSystem->A = malloc(sizeof(double*) * size);
    theSystem->B = malloc(sizeof(double) * size * (size+1));
    theSystem->A[0] = theSystem->B + size;
    theSystem->size = size;
    int i;
    for (i=1 ; i < size ; i++)
        theSystem->A[i] = theSystem->A[i-1] + size;
    return theSystem; }

femBandSystem *femBandSystemCreate(int size, int band) {
    femBandSystem *myBandSystem = malloc(sizeof(femBandSystem));
    myBandSystem->B = malloc(sizeof(double)*size*(band+1));
    myBandSystem->A = malloc(sizeof(double*)*size);
    myBandSystem->size = size;
    myBandSystem->band = band;
    myBandSystem->A[0] = myBandSystem->B + size;
    int i;
    for (i=1 ; i < size ; i++)
        myBandSystem->A[i] = myBandSystem->A[i-1] + band - 1;
    return(myBandSystem); }
```

1. Expliquer exactement comment l'allocation dynamique est effectuée dans ces deux fonctions ?
2. En particulier, observer comment la fonction `femBandSystemCreate` a été écrite afin que l'implémentation de l'élimination gaussienne et de l'assemblage dans un solveur bande sera particulièrement simple.

**16** Considérons le fragment de code informatique :

```
typedef struct {
    double *B;
    double **A;
    int size;
} femFullSystem;

typedef struct {
    femSolverType type;
    femFullSystem *local;
    void *solver;
} femSolver;

femSolver *femSolverCreate(int sizeLoc) {
    femSolver *mySolver = malloc(sizeof(femSolver));
    mySolver->local = femFullSystemCreate(sizeLoc);
    return (mySolver); }

femSolver *femSolverFullCreate(int size, int sizeLoc) {
    femSolver *mySolver = femSolverCreate(sizeLoc);
    mySolver->type = FEM_FULL;
    mySolver->solver = (femSolver *)femFullSystemCreate(size);
    return(mySolver); }

femSolver *femSolverBandCreate(int size, int sizeLoc, int band) {
    femSolver *mySolver = femSolverCreate(sizeLoc);
    mySolver->type = FEM_BAND;
    mySolver->solver = (femSolver *)femBandSystemCreate(size,band);
    return(mySolver); }
```

1. A quoi sert la structure `femSolver` ?
2. Que fait-on la fonction `femSolverCreate` ?
3. Peut-on remplacer le pointeur `void*` par un pointeur `femSolver*`
   dans la structure `femSolver` ?
   Comment faire ?
4. On dit que l'instruction `mySolver->solver = (femSolver *)femFullSystemCreate(size)`
   effectue un *casting* : expliquer :-)
5. Quel concept informatique implémente-t-on de manière assez primitive ici en `C` ?
6. Comment est-ce qu'on procéderait en `C++` ?