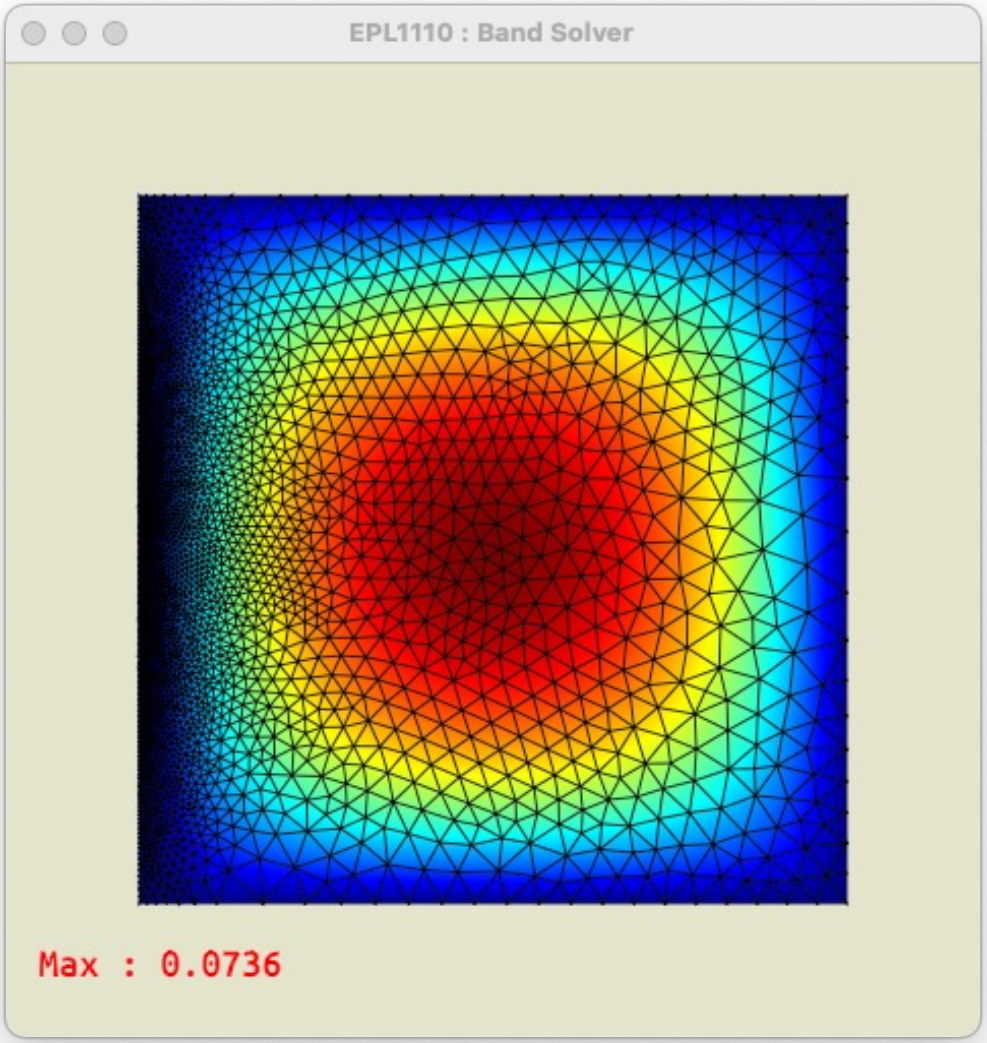



# Un petit mot sur la résolution de grands systèmes linéaires

# Résoudre un système triangulaire est facile…

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & & \dots & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

*Triangulation*

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ & a_{22} & \dots & a_{2n} \\ & & & \dots \\ & & & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

$$x_j = (b_j - \sum_{k=j+1}^{n} a_{jk} x_k) / a_{jj}$$

**Backward substitution**

# Triangularisation par élimination de Gauss

$$\begin{matrix} L1 \\ L2 \end{matrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$L2 = L2 - 4\,L1$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4-4 & 5-8 & 6-12 \\ \sim 0 & & \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} & \\ & \\ & \end{bmatrix}$$

*Karl Friedrich Gauss (1777-1855)*

# Triangularisation par élimination de Gauss

$$\begin{bmatrix} a_{11}^{(k)} & a_{12}^{(k)} & \dots & a_{1k}^{(k)} & \dots & a_{1n}^{(k)} \\ & a_{22}^{(k)} & \dots & a_{2k}^{(k)} & \dots & a_{2k}^{(k)} \\ & & & \dots & & \\ & & & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ & & & & \dots & \\ & & & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{bmatrix}$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)} \qquad i = k+1, \dots, n,\ j = k, \dots, n$$

$$b_{i}^{(k+1)} = b_{i}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} b_{k}^{(k)} \qquad i = k+1, \dots, n,$$

*Karl Friedrich Gauss (1777-1855)*

$$\underbrace{\begin{bmatrix} a_{11}^{(n)} & a_{12}^{(n)} & \dots & a_{1n}^{(n)} \\ & a_{22}^{(n)} & \dots & a_{2n}^{(n)} \\ & & \dots & \\ & & & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}}_{\mathbf{A}^{(n)}\mathbf{x}} = \underbrace{\begin{bmatrix} b_1^{(n)} \\ b_2^{(n)} \\ \dots \\ b_n^{(n)} \end{bmatrix}}_{\mathbf{b}^{(n)}}$$

# Le problème discret

Trouver $U_j \in \mathbb{R}^n$ tels que

$$\sum_{j=1}^{n} A_{ij} U_j \quad = \quad F_i, \qquad i = 1, n.$$

**Problème continu elliptique et linéaire**
**Méthode des éléments finis**
**Formulation de Galerkin**


**Matrice définie positive**

Trouver $U_j \in \mathbb{R}^n$ tels que

$$J(U_j) \quad = \quad \min_{V_j \in \mathbb{R}^n} \underbrace{\left( \sum_{i=1}^{n} \sum_{j=1}^{n} \tfrac{1}{2} V_i A_{ij} V_j - \sum_{i=1}^{n} V_i F_i \right)}_{J(V_j)},$$

# Aller simple vers le monde de l'algèbre linéaire…

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

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

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

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

**Utilisation des notations habituelles de l'algèbre linéaire**

# Deux grandes classes de méthodes de résolution

**Solveurs directs (Elimination gaussienne)**

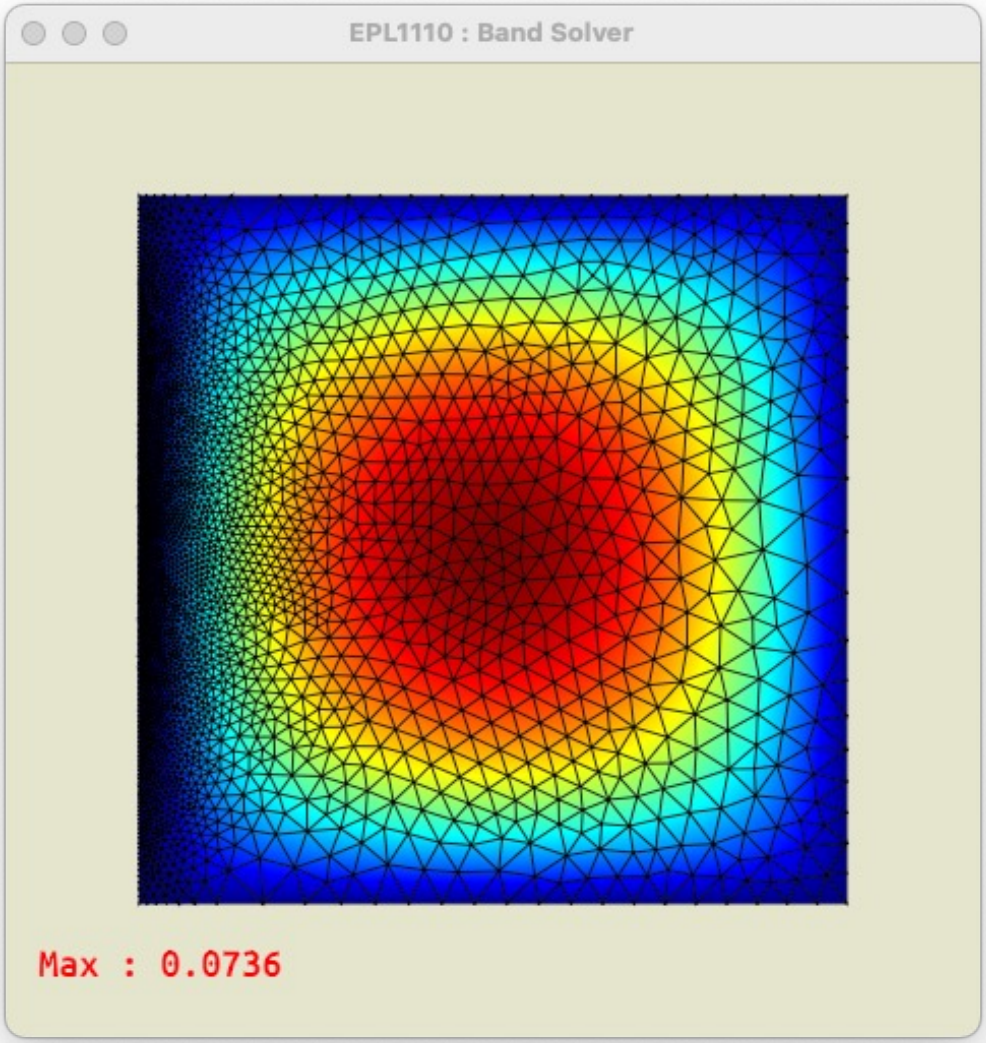
**Solveur de Gauss**
**Solveur de Gauss bande**
**Solveur de Gauss « skyline »**
**Solveur de Gauss frontal**
**Solveur de Gauss « Nested dissection »**

**Les méthodes de la plus grande pente et des gradients conjugés ne sont utilisables que pour une matrice définie positive**



**Solveurs itératifs**

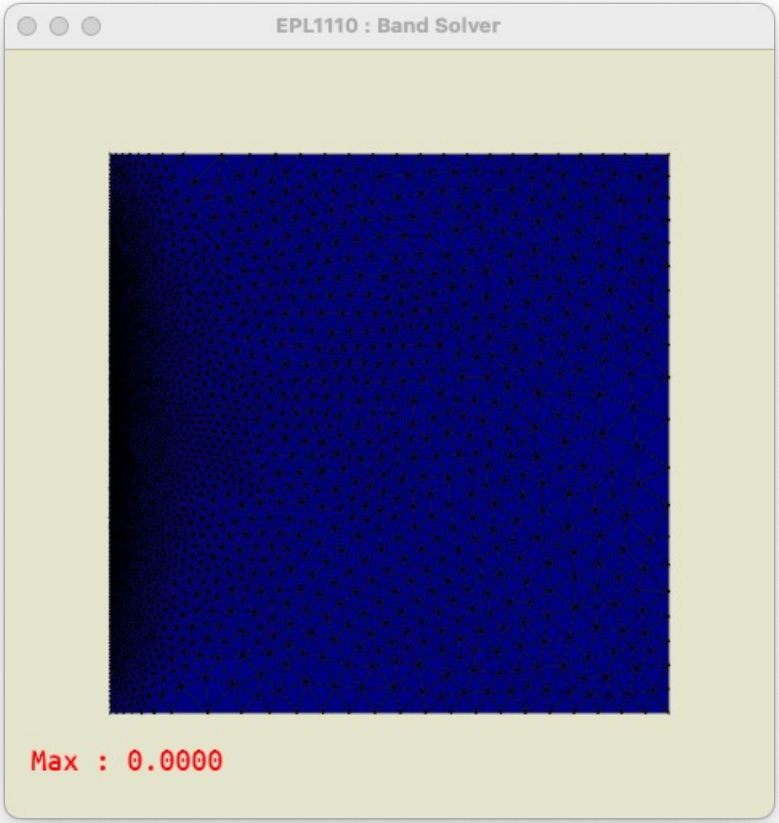
**Méthode de la plus grande pente**
**Méthode des gradients conjugés**
**Méthode de GMRES**

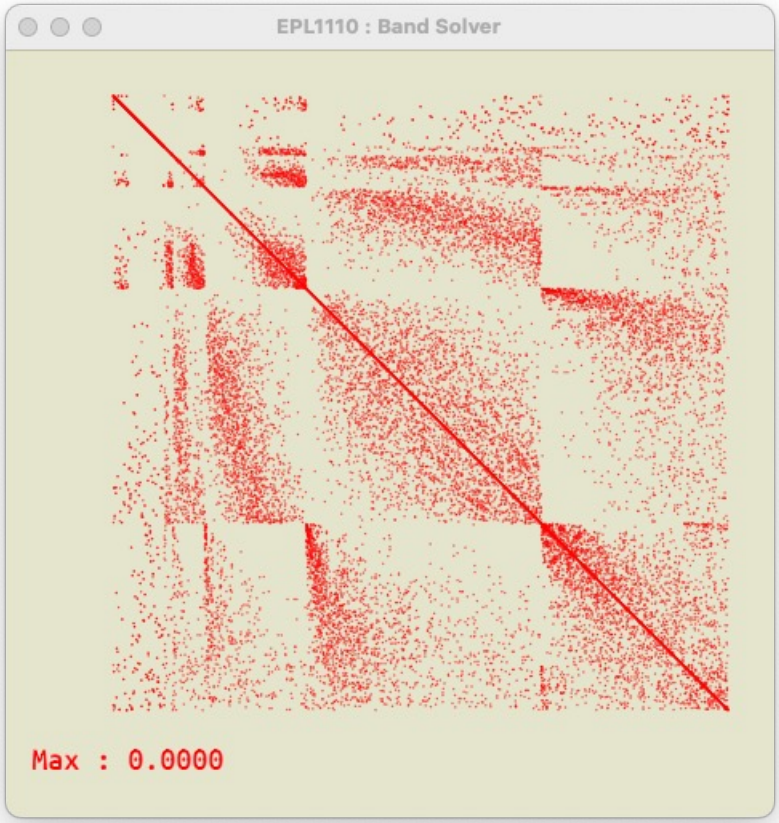



# Revenons à notre problème:-)

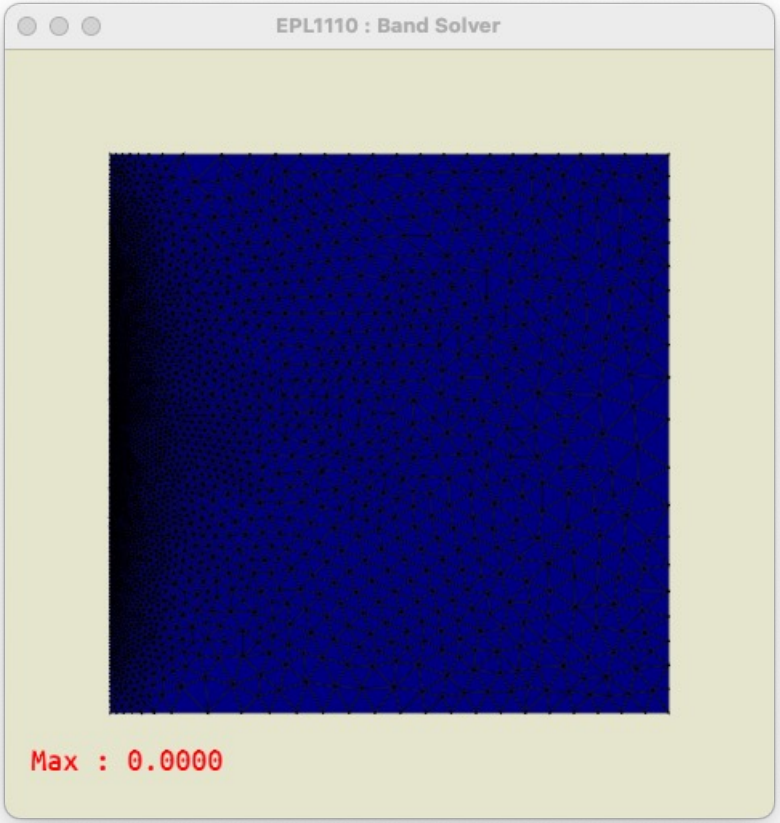
$$\nabla^2 u + 1 = 0 \quad \text{sur } \Omega$$

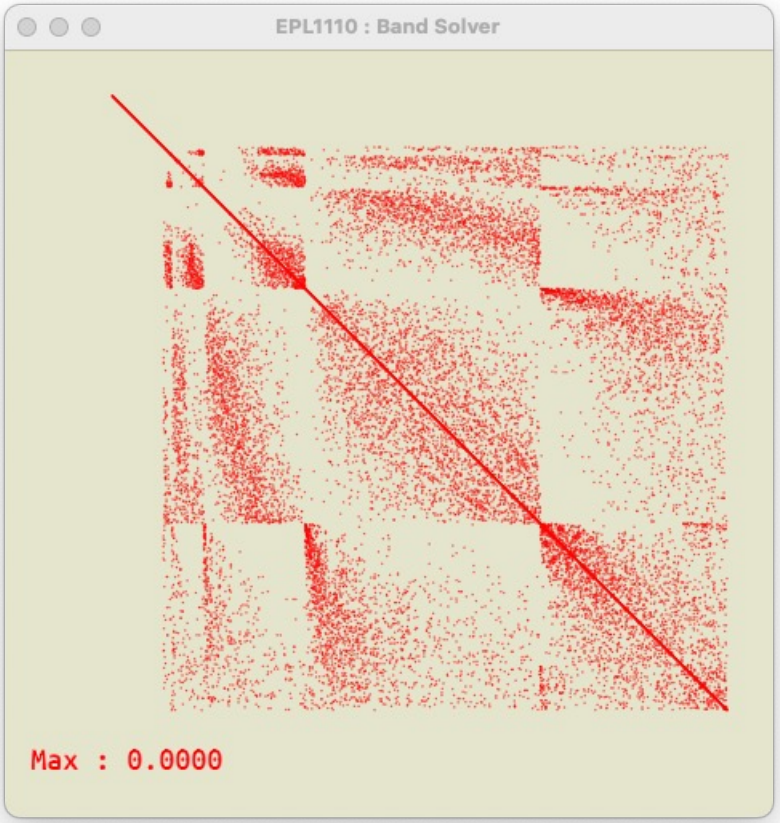
$$u = 0 \quad \text{sur } \partial\Omega$$



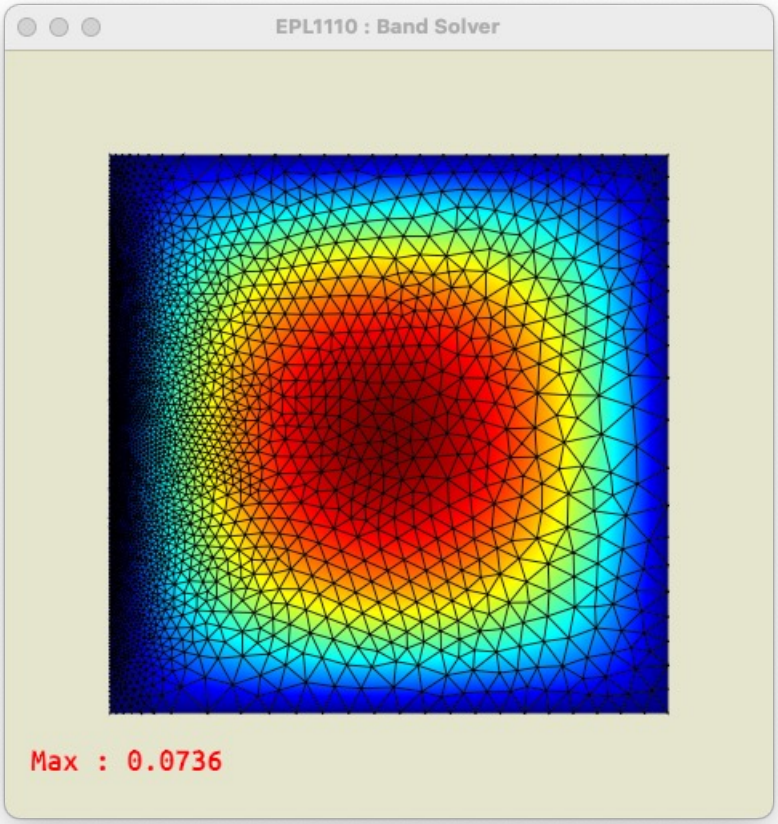



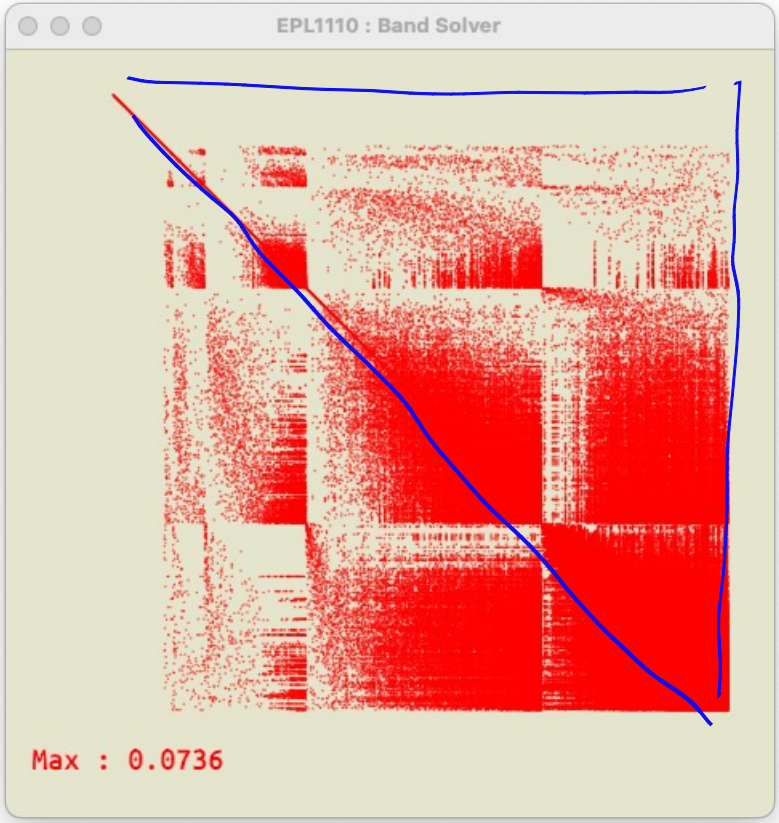

# Après l'assemblage :-)




Après les conditions frontières :-)




Après résolution :-)

# C'est facile à implémenter…

**Elimination effectuée « en place »**
**Ok, si tous les pivots sont non-nuls pendant les processus**

```
double  *matrixSolve(double **A, double *B, int size)

{    int i,j,k;


     /* Gauss elimination */
      for (k=0; k < size; k++) {
          if ( A[k][k] == 0 ) Error("zero pivot");
             for (i = k+1 ; i <  size; i++) {
                 factor = A[i][k] / A[k][k];
                 for (j = k+1 ; j < size; j++)
                      A[i][j] = A[i][j] - A[k][j] * factor;
                 B[i] = B[i] - B[k] * factor; }}

     /* Back-substitution */
     for (i = (size)-1; i >= 0 ; i--) {
           factor = 0;
           for (j = i+1 ; j < size; j++)
                 factor += A[i][j] * B[j];
            B[i] = ( B[i] - factor)/A[i][i]; }
     return(B);
}
```

**Oui, si la matrice est définie positive**

**Coût calcul : $O(n^3)$**

# Résolution par factorisation LU

**Factorisation de A en LU** *(effectuée par élimination gaussienne)*
**Résolution de Lz = b par substitution directe**
**Résolution de Ux = z par substitution arrière**

$$l_{ik} = -\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \qquad i = k+1, \ldots, n,\ k = 1, \ldots, n$$

$$l_{ii} = 1 \qquad i = 1, \ldots, n$$

**Même coût qu'une élimination gaussienne,**
**Il suffit juste de conserver L**
**Très utile, si on souhaite traiter plusieurs cas de charges !**
**Mais plus gourmand en mémoire : facteur critique..**

# Matrice symétrique définie-positive : Méthode de Cholesky

$$
\begin{aligned}
b_{11} &= \sqrt{a_{11}} \\
b_{i1} &= \frac{a_{i1}}{b_{11}} && i = 2, \ldots, n \\
b_{jj} &= \sqrt{a_{jj} - \sum_{k=1}^{j-1} b_{jk}^2} && j = 2, \ldots, n \\
b_{ij} &= \frac{a_{ij} - \sum_{k=1}^{j-1} b_{ik} b_{jk}}{b_{jj}} && j = 2, \ldots, n,\ i = j+1, \ldots, n
\end{aligned}
$$

$$\mathbf{A = BB^T}$$

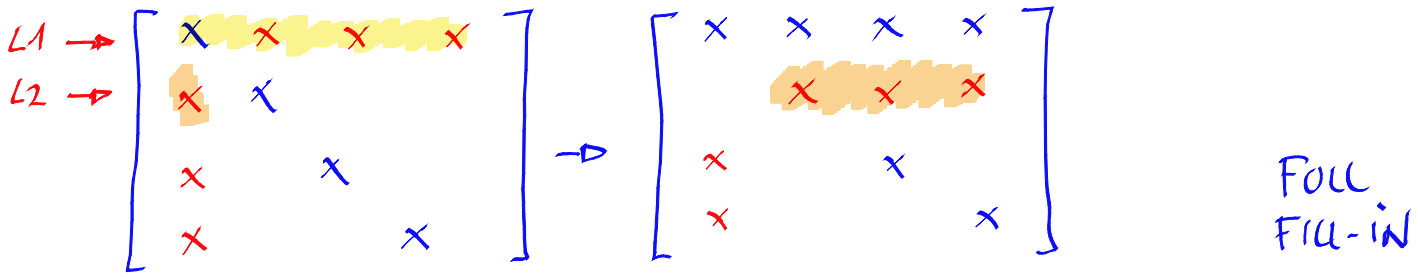
**Plus intéressante que l'élimination gaussienne en termes d'opération,**
**Mais valable que pour des matrices symétriques définies positives**

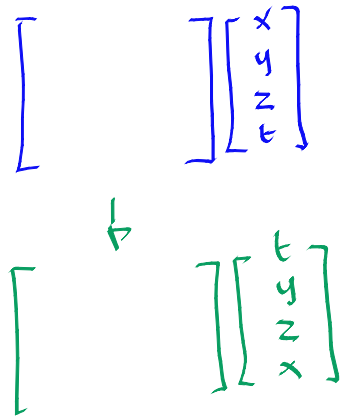
# Factorisation de matrices creuses : Problème du *fill-in*

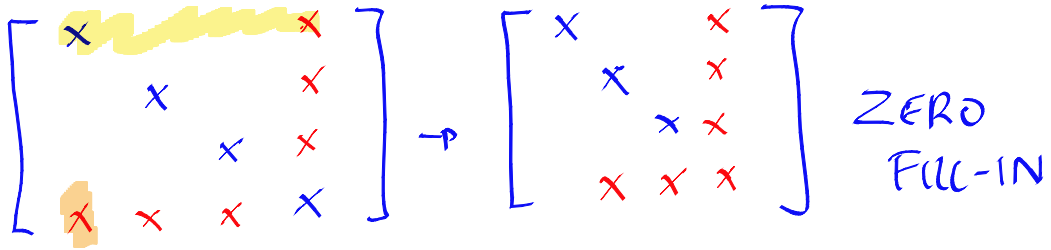
$$\mathbf{A} = \begin{bmatrix} \lozenge & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge \\ \lozenge & \lozenge & & & & & \\ \lozenge & & \lozenge & & & & \\ \lozenge & & & \lozenge & & & \\ \lozenge & & & & \lozenge & & \\ \lozenge & & & & & \lozenge & \\ \lozenge & & & & & & \lozenge \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} \lozenge & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge \\ & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge \\ & & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge \\ & & & \lozenge & \lozenge & \lozenge & \lozenge \\ & & & & \lozenge & \lozenge & \lozenge \\ & & & & & \lozenge & \lozenge \\ & & & & & & \lozenge \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} \lozenge & & & & & & \lozenge \\ & \lozenge & & & & & \lozenge \\ & & \lozenge & & & & \lozenge \\ & & & \lozenge & & & \lozenge \\ & & & & \lozenge & & \lozenge \\ & & & & & \lozenge & \lozenge \\ \lozenge & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} \lozenge & & & & & & \lozenge \\ & \lozenge & & & & & \lozenge \\ & & \lozenge & & & & \lozenge \\ & & & \lozenge & & & \lozenge \\ & & & & \lozenge & & \lozenge \\ & & & & & \lozenge & \lozenge \\ & & & & & & \lozenge \end{bmatrix}$$

# Fill-in

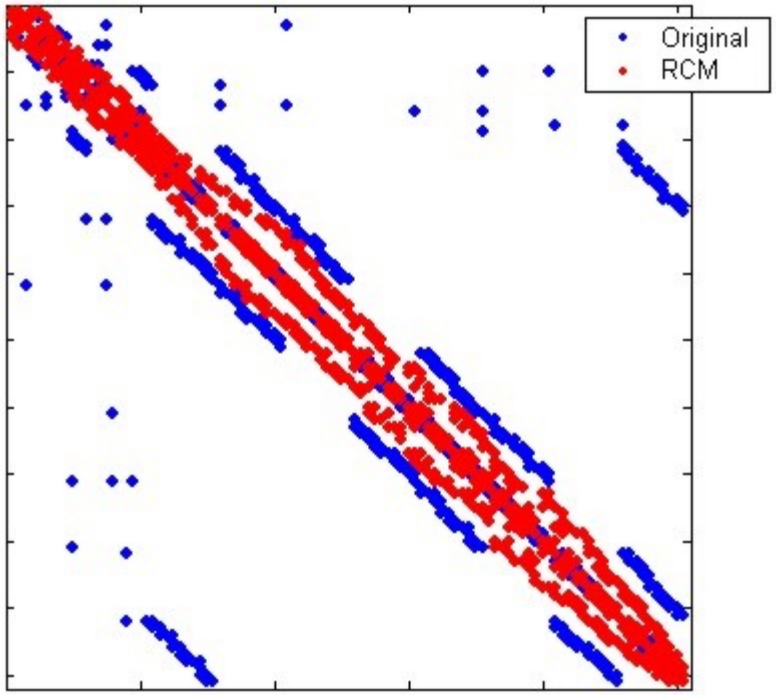


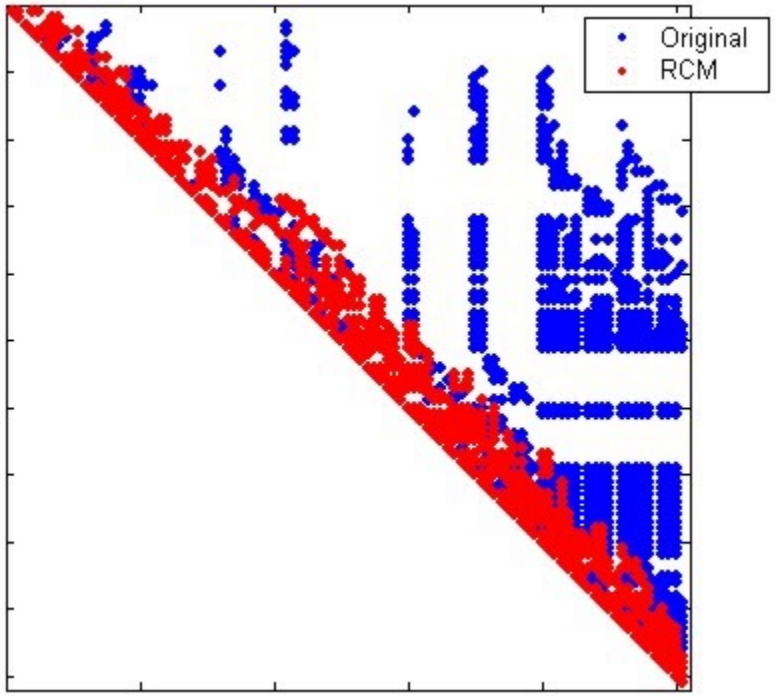





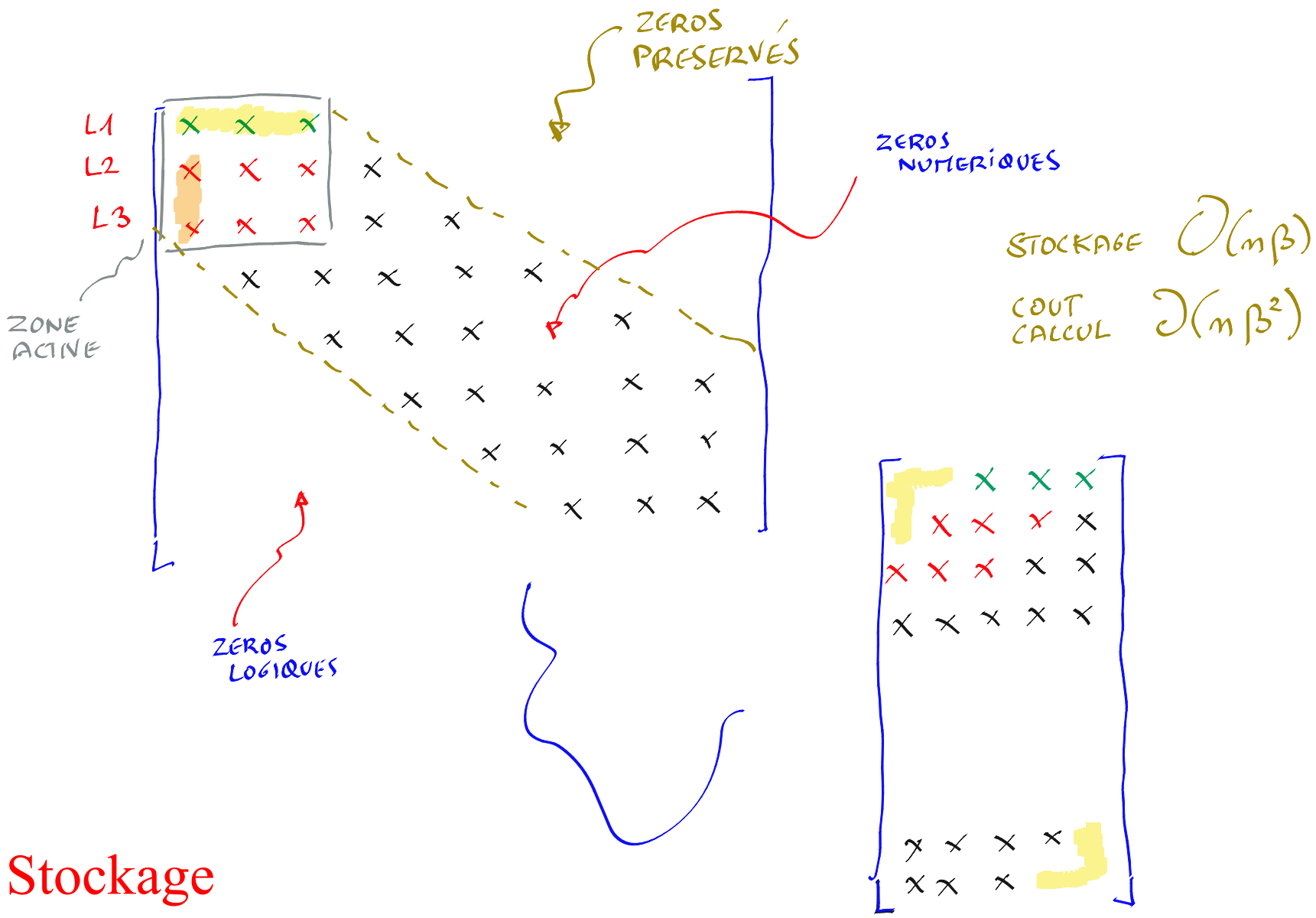

**Seule, la numérotation des noeuds est cruciale !**

# *Fill-in* et renumérotation des variables !



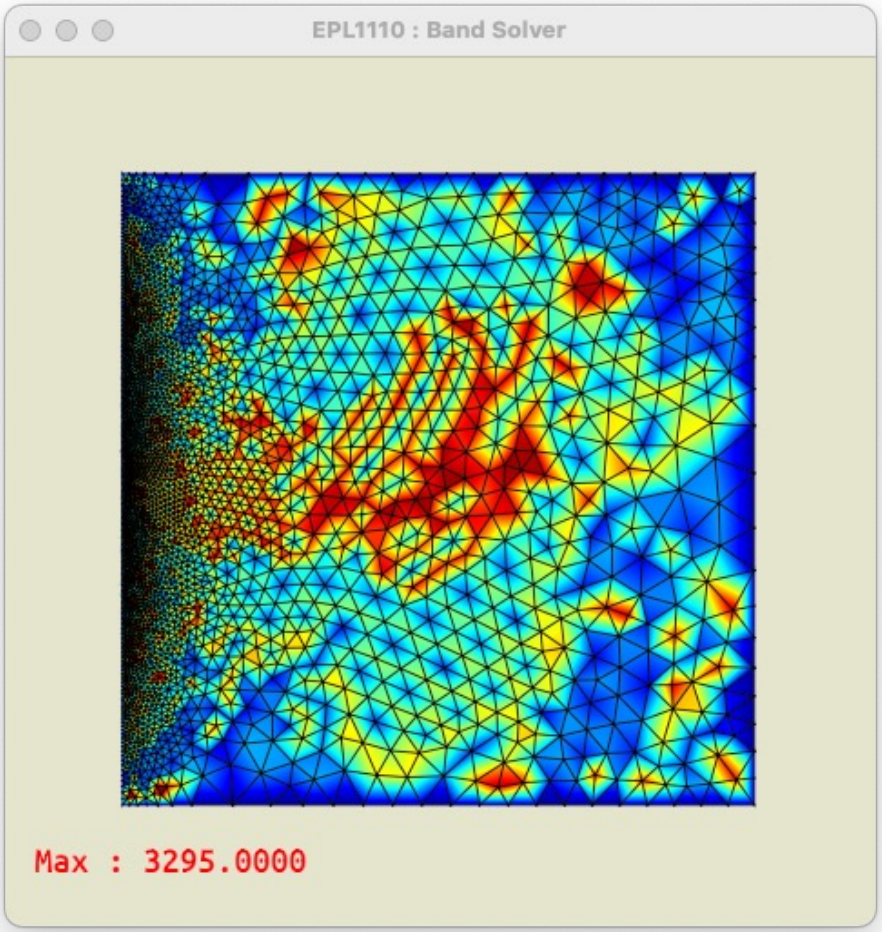



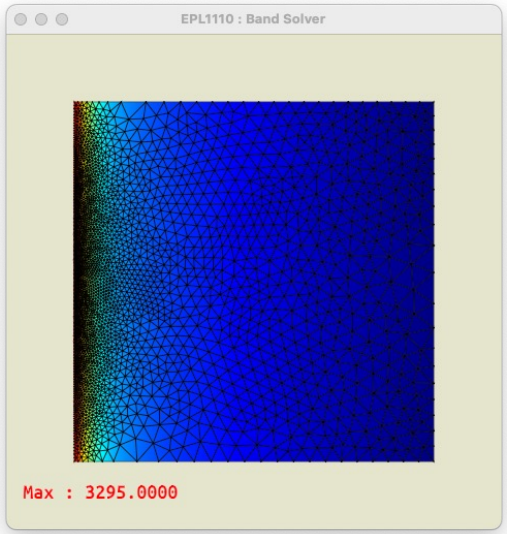

***Yannakakis, Computing the minimum fill-in is NP-complete, SIAM J. Algebraic and Discrete Methods, Vol. 2, 1981, 77-79.***




# Stockage de la matrice bande

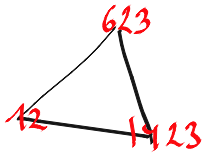
# En modifiant la numérotation des noeuds…

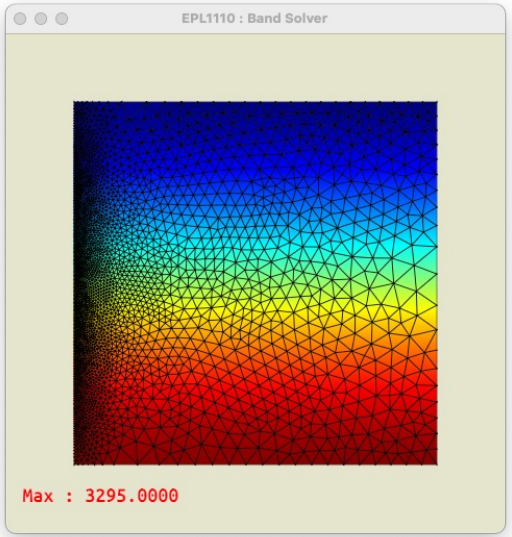


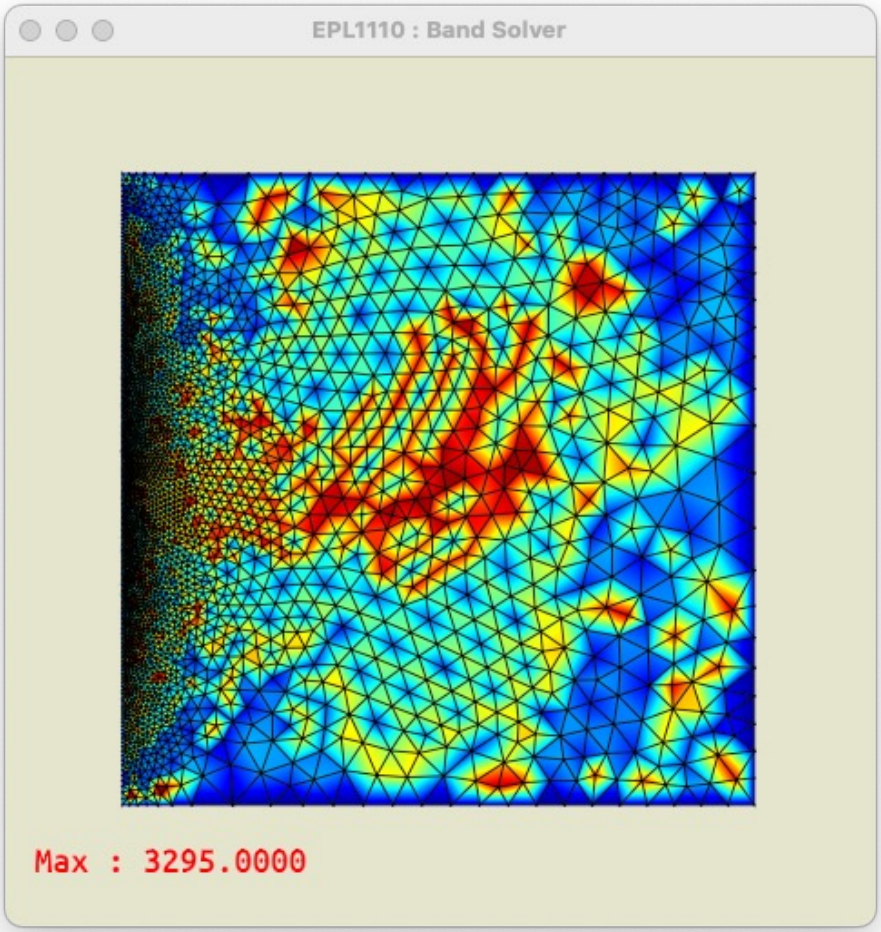




```
    BandSize Gaussian
    Matrix size       :      3296
    Matrix band       :       610
    Bytes required    : 16110848


    Full Gaussian elimination
    Matrix size       :      3296
    Bytes required    : 86935296
```

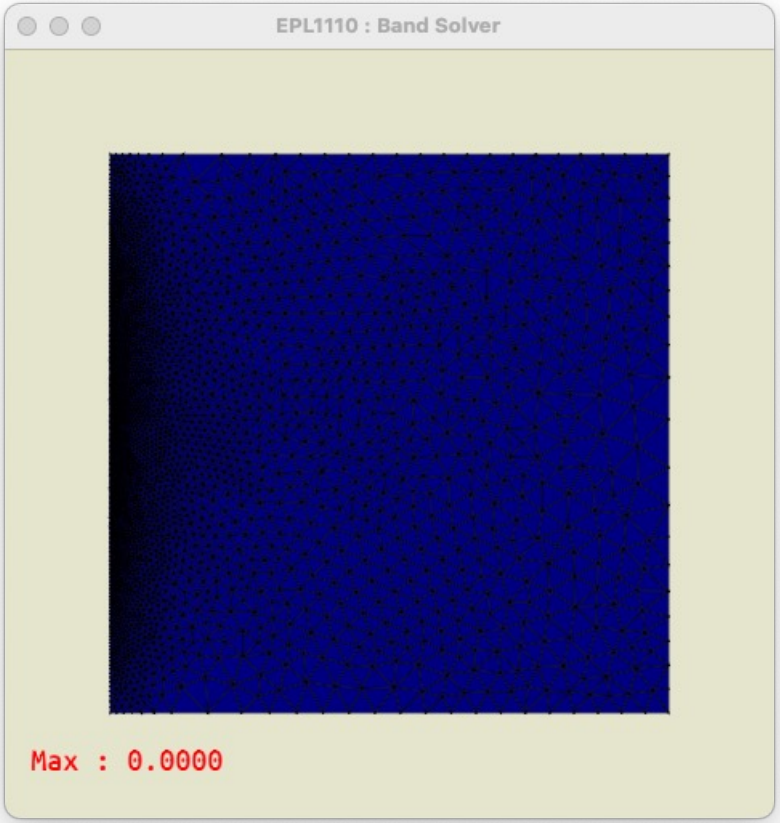
# En modifiant la numérotation des noeuds…

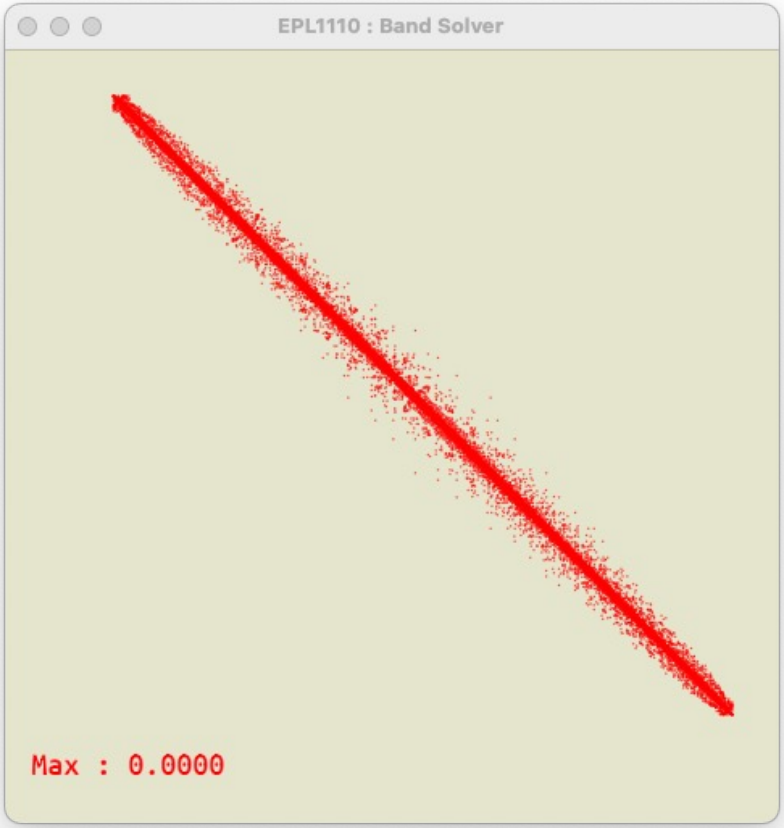


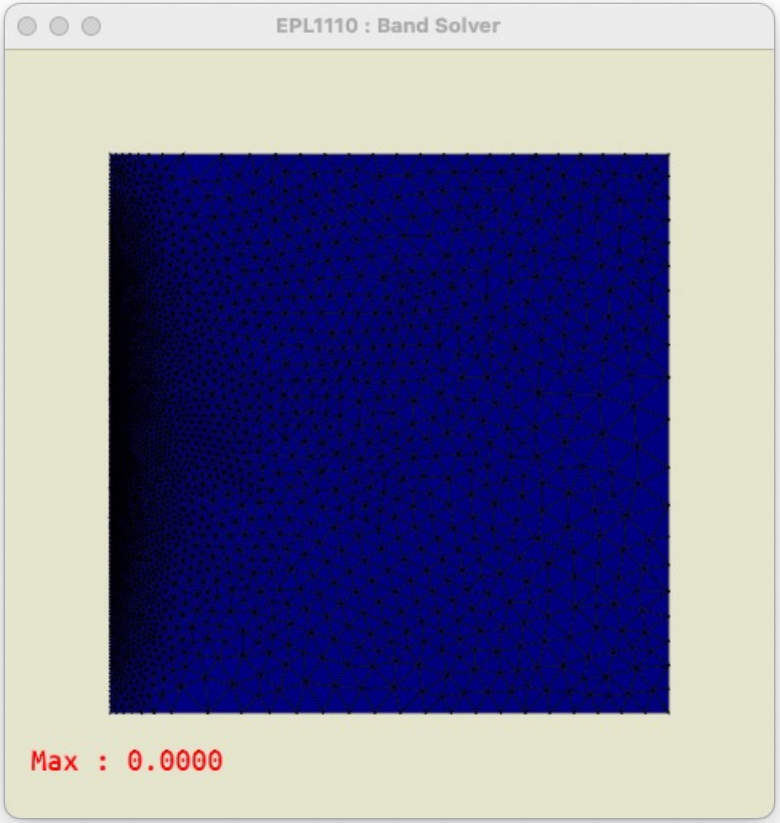



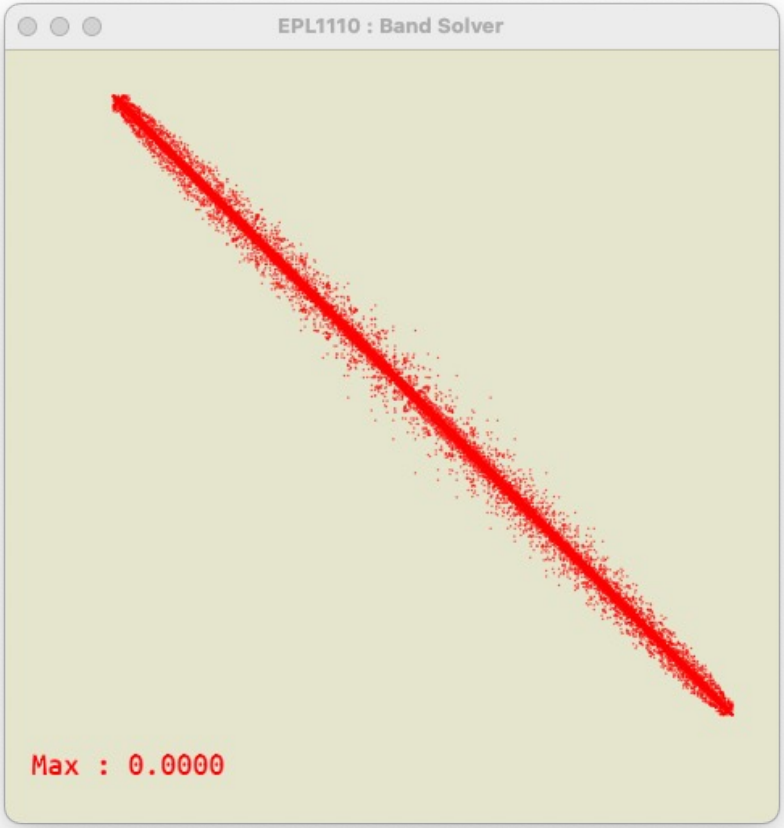

```
BandSize Gaussian
Matrix size       :      3296
Matrix band       :       402
Bytes required    : 10626304



Full Gaussian elimination
Matrix size       :      3296
Bytes required    : 86935296
```



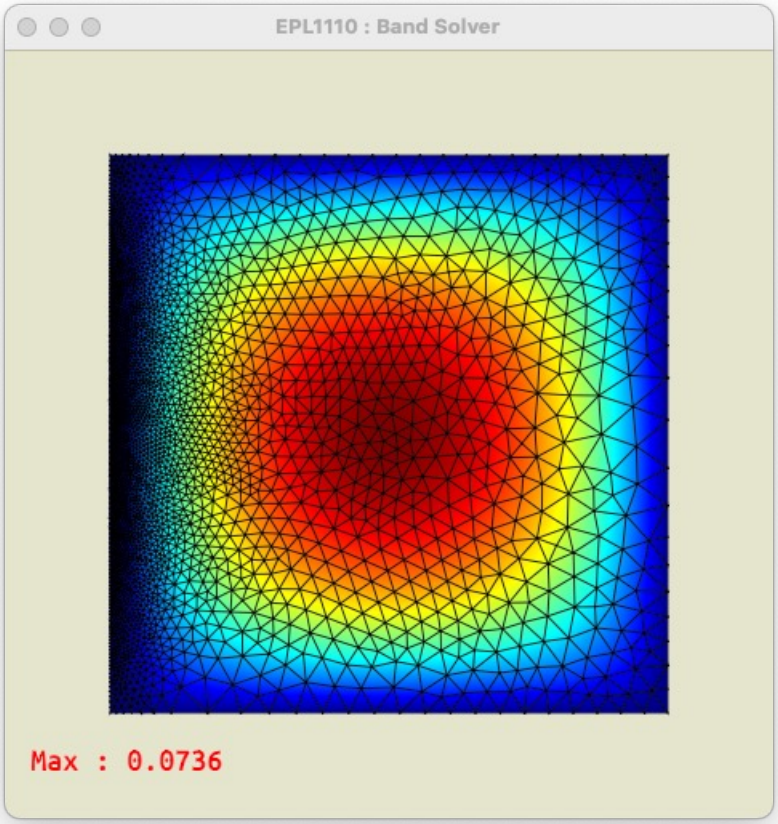



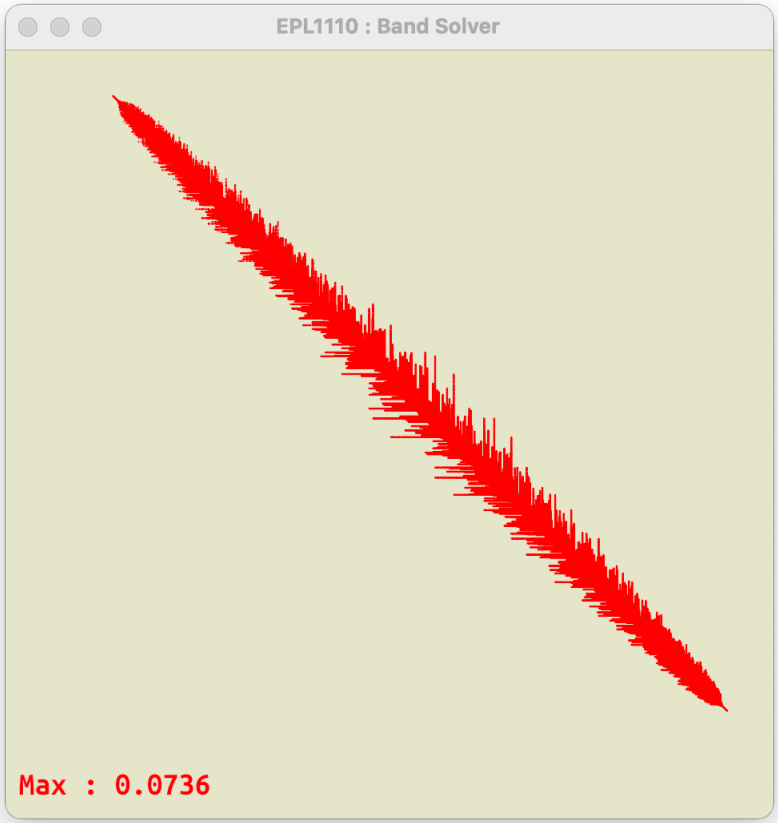

Après l’assemblage :-)




# Après les conditions frontières :-)

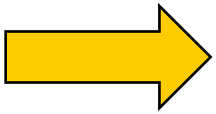


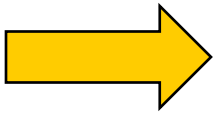

Après résolution :-)

# Quel est le vrai facteur critique pour des simulations de grande taille ?

**Précision du résultat**
**Faut-il pivoter ?**
**Comment minimiser la propagation des erreurs d'arrondis ?**
**Simple ou double précision ?**
**Matrices sym. déf. pos. : il n'est pas requis de pivoter !**

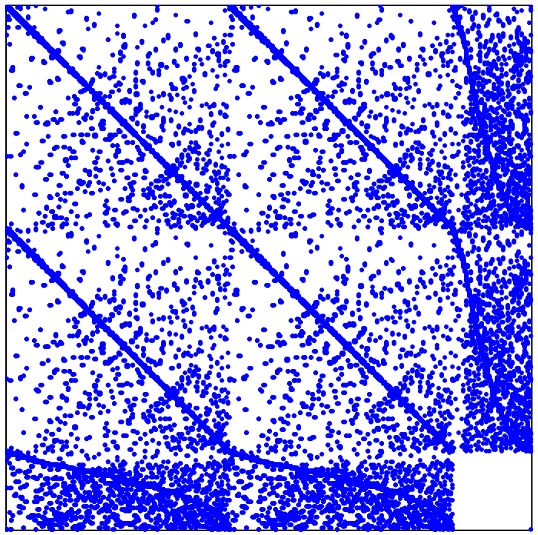
**Aujourd'hui** 

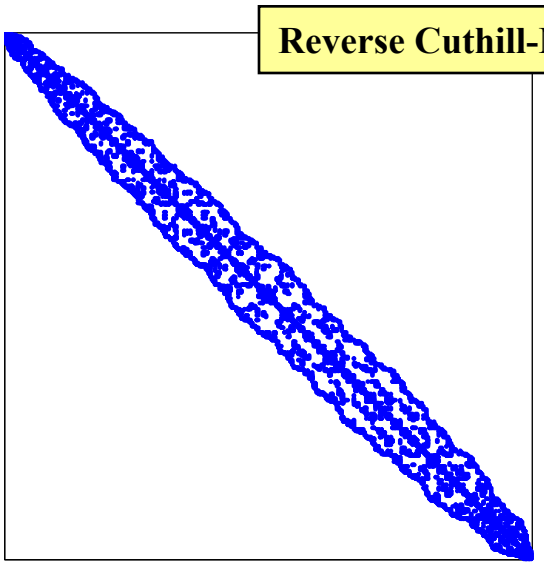
**Espace mémoire requis**
**Stockage de U ou LU**
**Calcul en simple ou en double précision ?**
**Stockage en simple ou en double précision ?**

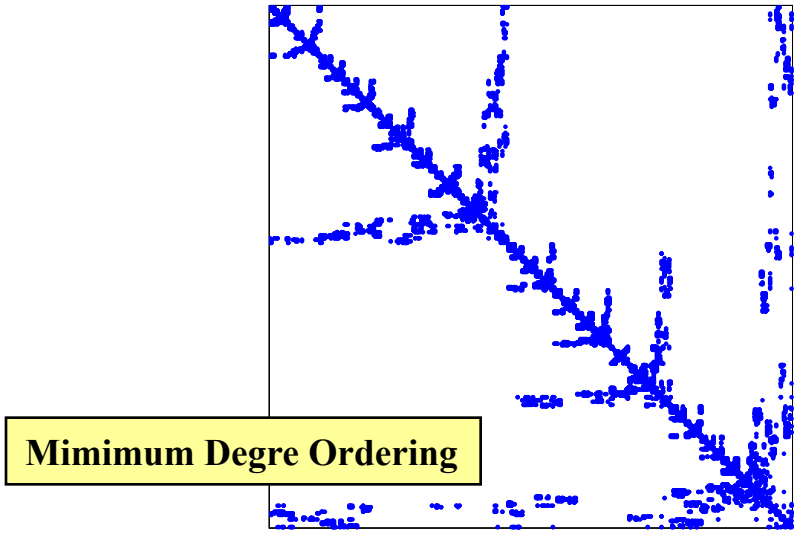
**Hier** 

**Temps de calcul « cpu »**
**Heuristiques de rénumérotation**
**Utilisation des outils BLAS et LAPACK**

# Heuristiques de renumérotation





# Solveur de Gauss pour matrices bandes

**Coût calcul : $O(n\beta^2)$**

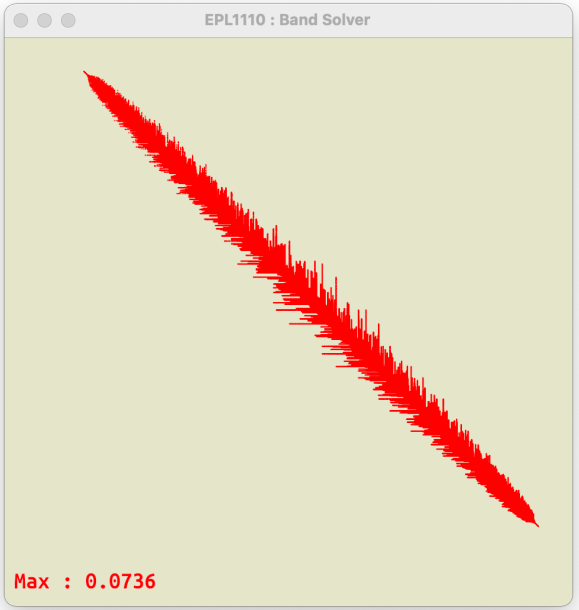
$$\beta(A_{ij}) - 1 = max_{ij}\left\{|i-j|, \quad \forall (i,j) \text{ tels que } A_{ij} \neq 0\right\}.$$

Concept de largeur de bande

$$\mathbf{A} = \begin{bmatrix} \Diamond & \Diamond & \Diamond & & & & & \\ \Diamond & \Diamond & & \Diamond & & & & \\ \Diamond & & \Diamond & \Diamond & \Diamond & & & \\ & \Diamond & \Diamond & \Diamond & & \Diamond & & \\ & & \Diamond & & \Diamond & \Diamond & \Diamond & \\ & & & \Diamond & \Diamond & \Diamond & & \\ & & & & \Diamond & & \Diamond & \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} \Diamond & \Diamond & \Diamond & & & & & \\ & \Diamond & \Diamond & \Diamond & & & & \\ & & \Diamond & \Diamond & \Diamond & & & \\ & & & \Diamond & \Diamond & \Diamond & & \\ & & & & \Diamond & \Diamond & \Diamond & \\ & & & & & \Diamond & \Diamond & \\ & & & & & & \Diamond & \end{bmatrix}$$

**La numérotation des inconnues est cruciale**

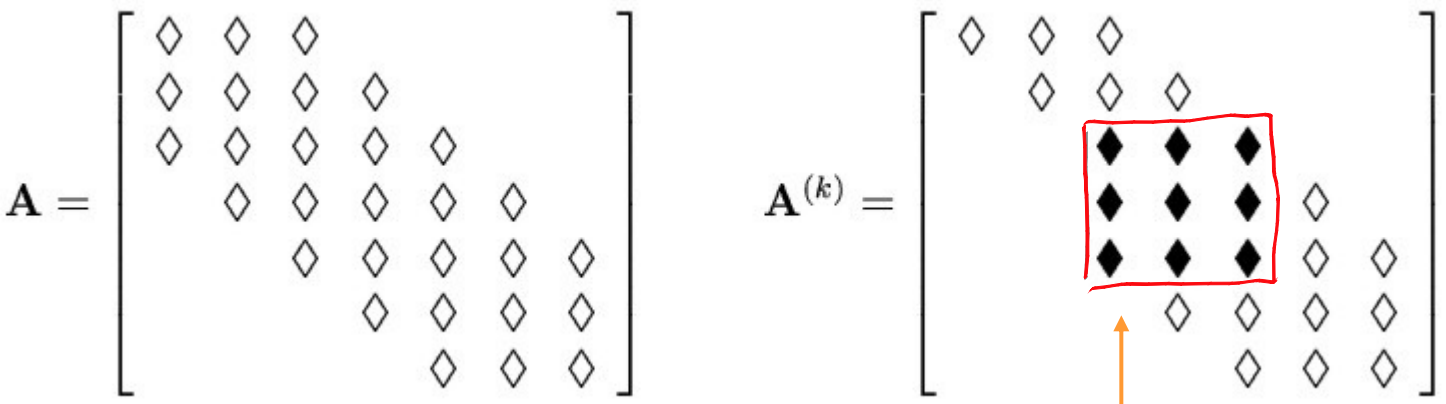
**Zéro logique traité comme un zéro numérique**




```
femBandSystem *femBandSystemCreate(int size,int band);
void femBandSystemInit(Band *myBand);
void femBandSystemPrint(Band *myBand);
void femBandSystemPrintInfos(Band *myBand);
void femBandSystemConstrain(Band *myBand,int myNode,double myValue);
void femBandSystemdEliminate(Band *myBand);
void femBandSystemAssemble(Band *myBand,double *A,double *B,int *row,int *col,int n);
```

# Homework 5 :
# un solveur creux bande !

# Solveur de Gauss frontal



**Coût calcul : $O(n\beta^2)$**

**Matrice active stockée sur la mémoire à accès rapide**

**Le reste est stocké sur la mémoire secondaire**

**La méthode frontale est cependant plus générale et peut s'appliquer à une matrice non-bande !**
**On effectue l'élimination et l'assemblage de manière simultanée :**

**ce sera la numérotation des éléments qui sera critique**

# Solveur de Gauss frontal

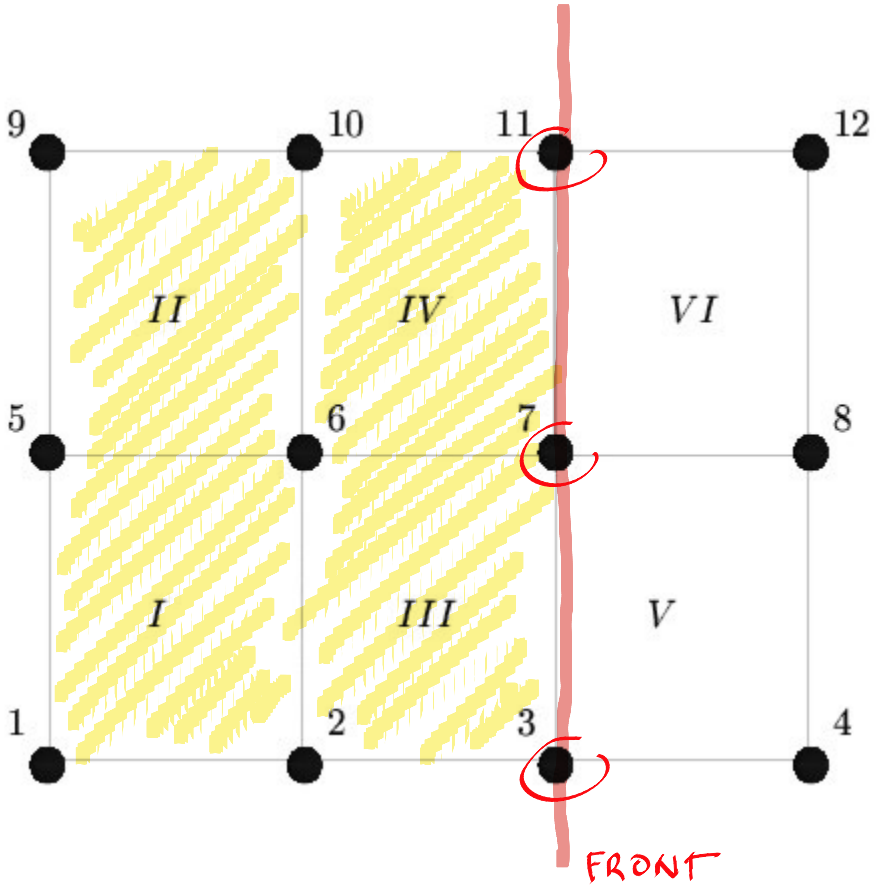
$$\mathbf{A} = \begin{bmatrix} \lozenge & \lozenge & \lozenge & & & & \\ \lozenge & \lozenge & \lozenge & \lozenge & & & \\ \lozenge & \lozenge & \lozenge & \lozenge & \lozenge & & \\ & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge & \\ & & \lozenge & \lozenge & \lozenge & \lozenge & \lozenge \\ & & & \lozenge & \lozenge & \lozenge & \lozenge \\ & & & & \lozenge & \lozenge & \lozenge \end{bmatrix}$$

$$\mathbf{A}^{(k)} = \begin{bmatrix} \lozenge & \lozenge & \lozenge & & & & \\ & \lozenge & \lozenge & \lozenge & & & \\ & & \blacklozenge & \blacklozenge & \blacklozenge & & \\ & & \blacklozenge & \blacklozenge & \blacklozenge & \lozenge & \\ & & \blacklozenge & \blacklozenge & \blacklozenge & \lozenge & \lozenge \\ & & & \lozenge & \lozenge & \lozenge & \lozenge \\ & & & & \lozenge & \lozenge & \lozenge \end{bmatrix}$$

$$\mathbf{A}^{(k)} = \begin{bmatrix} \lozenge & \lozenge & \lozenge & & & & \\ & \lozenge & \lozenge & \lozenge & & & \\ & & \blacklozenge & \blacklozenge & \blacklozenge & & \\ & & \blacklozenge & \blacklozenge & \blacklozenge & \lozenge & \\ & & \blacklozenge & \blacklozenge & \blacklozenge & \lozenge & \lozenge \\ & & & \lozenge & \lozenge & \lozenge & \lozenge \\ & & & & \lozenge & \lozenge & \lozenge \end{bmatrix}$$

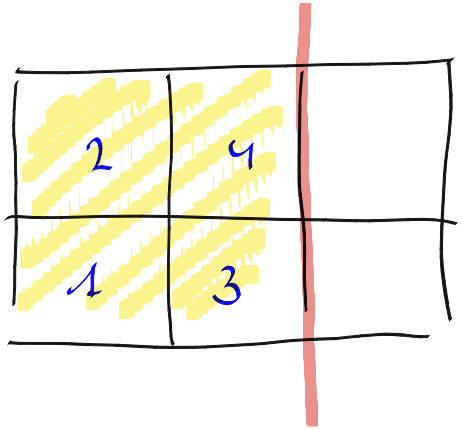
# Exemple

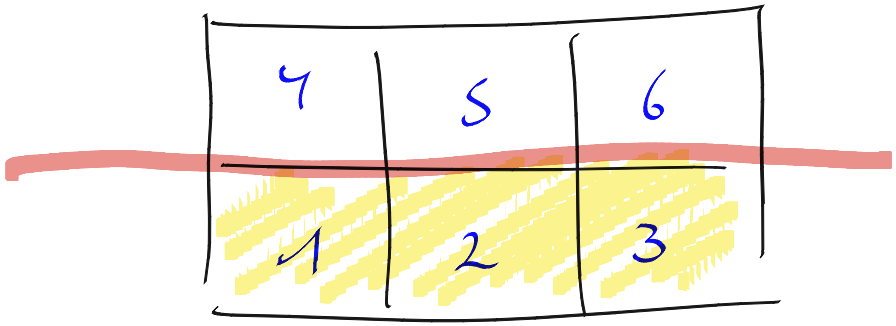
**Seule, la numérotation des éléments est cruciale !**




| Elément | Sommets | | | |
|---|---|---|---|---|
| 1 | 1 | 2 | 6 | 5 |
| 2 | 5 | 6 | 10 | 9 |
| 3 | 2 | 3 | 7 | 6 |
| 4 | 11 | 10 | 6 | 7 |
| 5 | 3 | 4 | 8 | 7 |
| 6 | 7 | 8 | 12 | 11 |

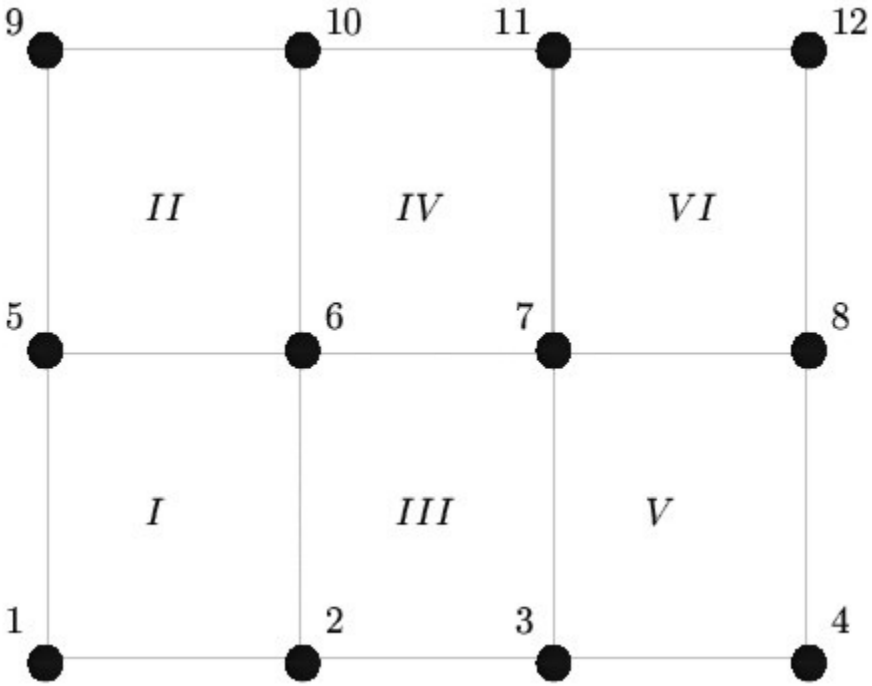
# Exemple



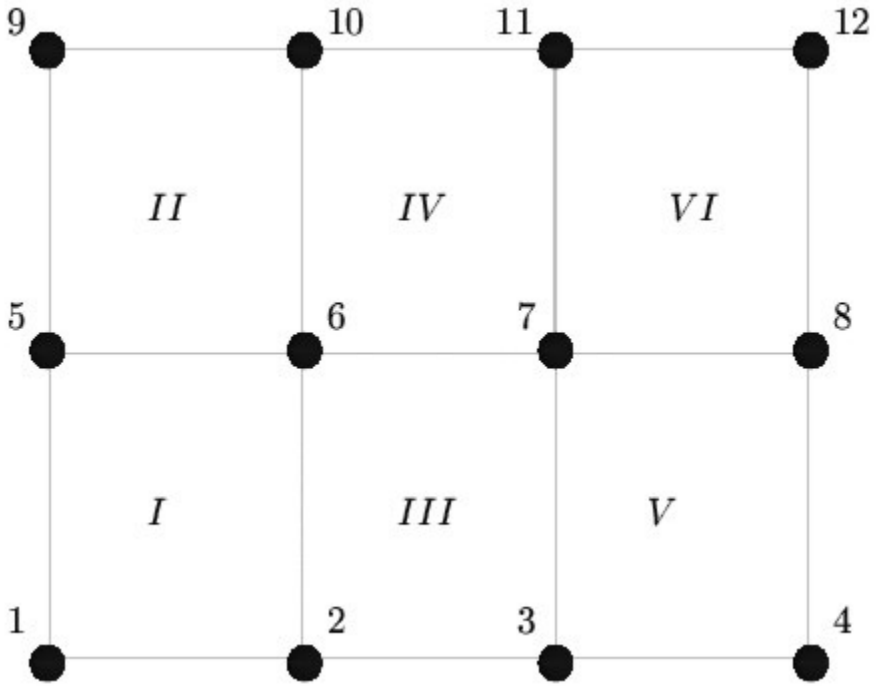

**Seule, la numérotation des éléments est cruciale !**

# L'assemblage est un processus séquentiel

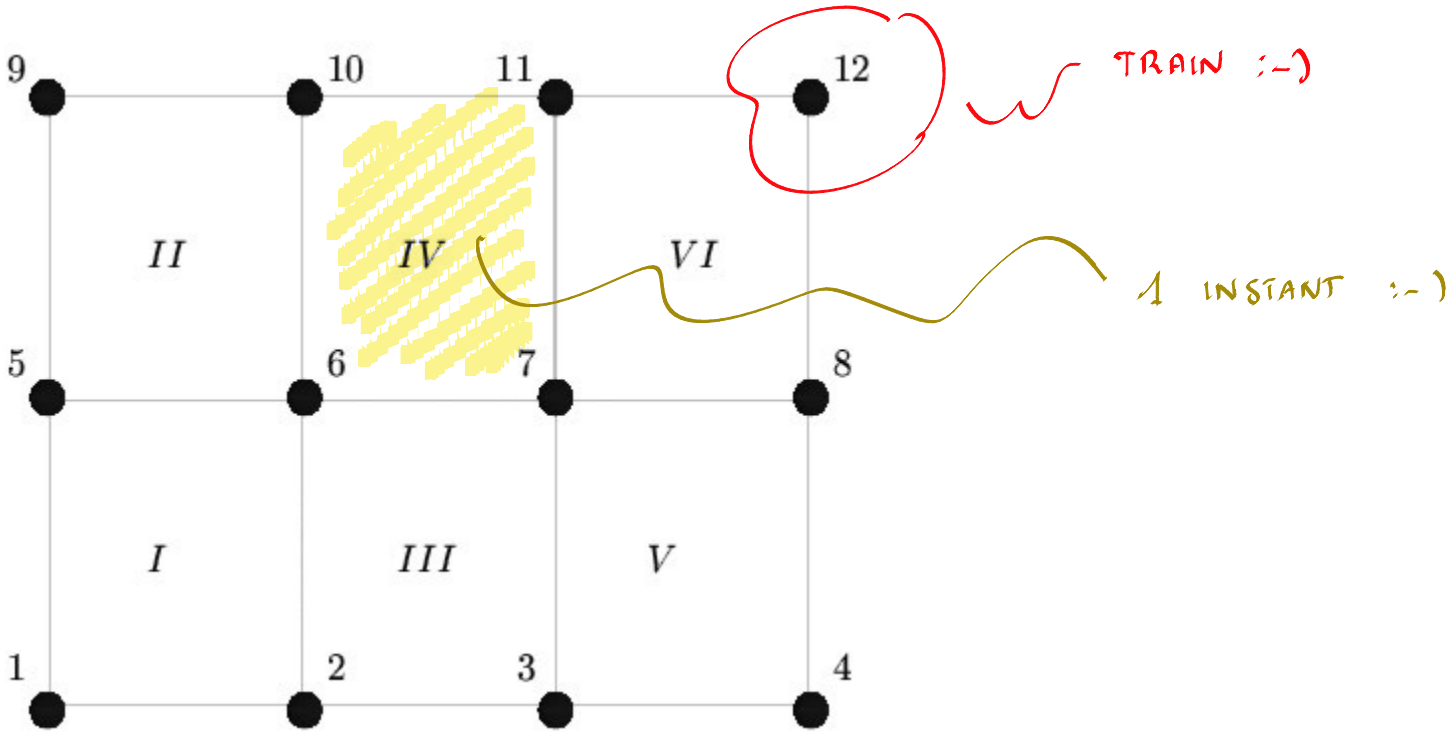



ELEMENT 1 = 12h JOUR 1
2 = 13h JOUR 2
3 = 14h JOUR 3
4 = 15h JOUR 4
5 = 16h JOUR 5
6 = 17h JOUR 6

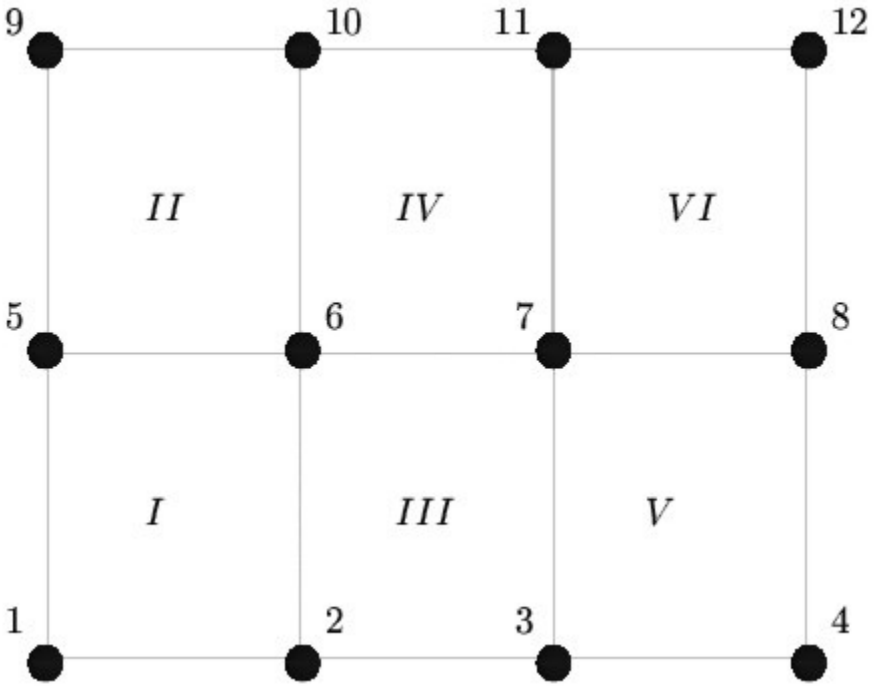
# Chaque élément est une heure !

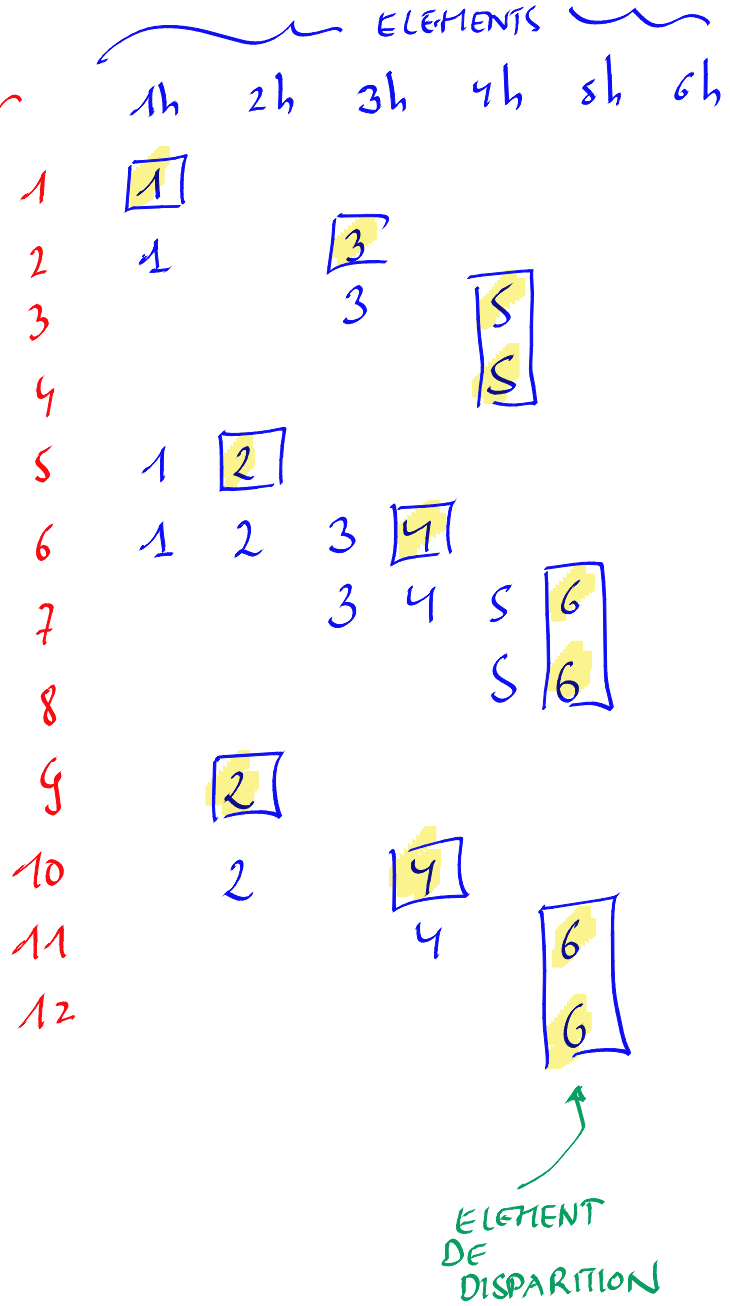


# Chaque nœud est un train !

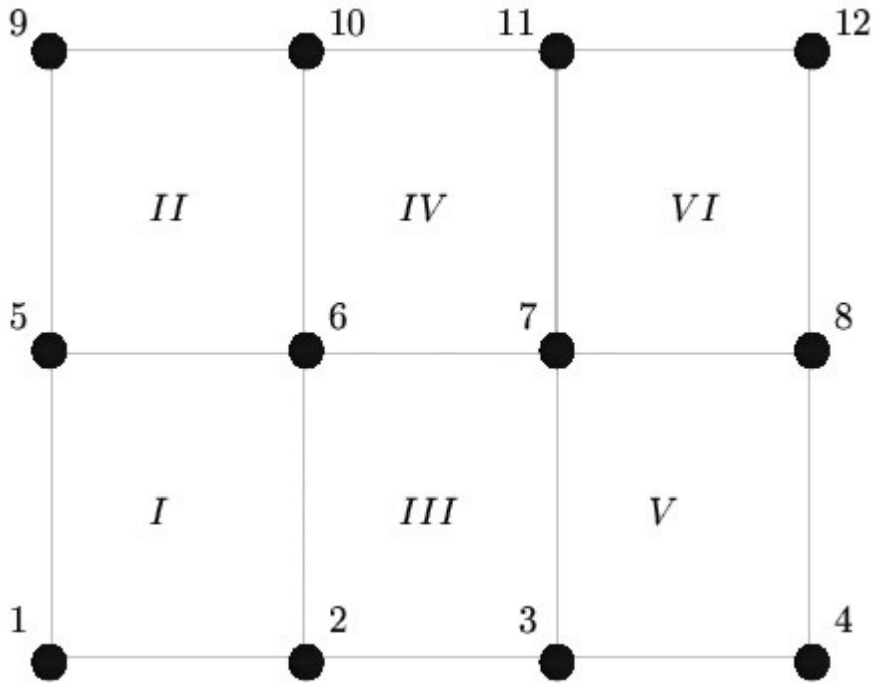
# Le train est passé !



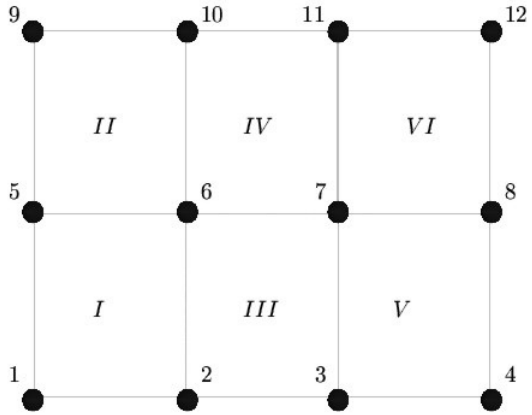



# Heure ou élément où le train est passé !

| Elément | Sommets | | | |
|---|---|---|---|---|
| 1 | 1 | 2 | 6 | 5 |
| 2 | 5 | 6 | 10 | 9 |
| 3 | 2 | 3 | 7 | 6 |
| 4 | 11 | 10 | 6 | 7 |
| 5 | 3 | 4 | 8 | 7 |
| 6 | 7 | 8 | 12 | 11 |



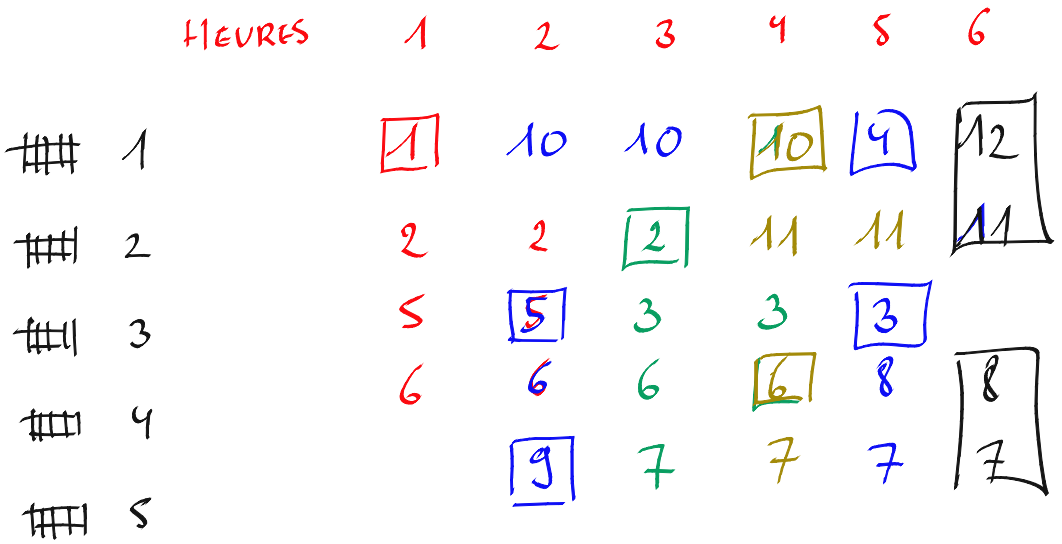
# Calcul de l'élément de disparition




| Sommet | Elément de disparition |
|---|---|
| 1 | [1] |
| 2 | 1 [3] |
| 3 | 3 [5] |
| 4 | [5] |
| 5 | 1 [2] |
| 6 | 1 2 3 [4] |
| 7 | 3 4 5 [6] |
| 8 | 5 [6] |
| 9 | [2] |
| 10 | 2 [4] |
| 11 | 4 [6] |
| 12 | [6] |

# Attribuer une voie libre à un train !

| Elément | Sommets | | | |
|---|---|---|---|---|
| 1 | 1 | 2 | 6 | 5 |
| 2 | 5 | 6 | 10 | 9 |
| 3 | 2 | 3 | 7 | 6 |
| 4 | 11 | 10 | 6 | 7 |
| 5 | 3 | 4 | 8 | 7 |
| 6 | 7 | 8 | 12 | 11 |



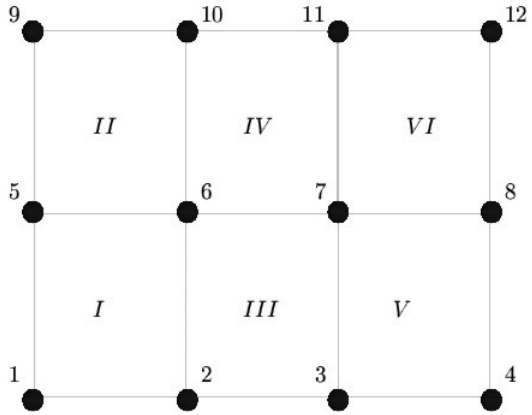

| Sommet | Elément de disparition | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | | | | | |
| 2 | 1 | | 3 | | | |
| 3 | | | 3 | | 5 | |
| 4 | | | | | 5 | |
| 5 | 1 | 2 | | | | |
| 6 | 1 | 2 | 3 | 4 | | |
| 7 | | | 3 | 4 | 5 | 6 |
| 8 | | | | | 5 | 6 |
| 9 | | 2 | | | | |
| 10 | | 2 | | 4 | | |
| 11 | | | | 4 | | 6 |
| 12 | | | | | | 6 |

# Attribuer une voie libre à un train !

| | Occupation de la matrice active lors de l'assemblage des éléments | | | | | |
|---|---|---|---|---|---|---|
| Elément | 1 | 2 | 3 | 4 | 5 | 6 |
| Destination | | | | | | |
| 1 | **1** | 10 | 10 | **10** | **4** | **12** |
| 2 | 2 | 2 | **2** | 11 | 11 | **11** |
| 3 | 6 | 6 | 6 | **6** | 8 | **8** |
| 4 | 5 | **5** | 3 | 3 | **3** | |
| 5 | | **9** | 7 | 7 | 7 | **7** |
| 6 | | | | | | |

→ **Calcul de la taille requise pour la matrice active**
**Comment dimensionner la jonction Nord-Midi ?**

# Factorisation symbolique




| | Occupation de la matrice active lors de l'assemblage des éléments | | | | | |
|---|---|---|---|---|---|---|
| Elément | 1 | 2 | 3 | 4 | 5 | 6 |
| Destination | | | | | | |
| 1 | **1** | 10 | 10 | **10** | **4** | **12** |
| 2 | 2 | 2 | **2** | 11 | 11 | **11** |
| 3 | 6 | 6 | 6 | **6** | 8 | **8** |
| 4 | 5 | **5** | 3 | 3 | **3** | |
| 5 | | **9** | 7 | 7 | 7 | **7** |
| 6 | | | | | | |

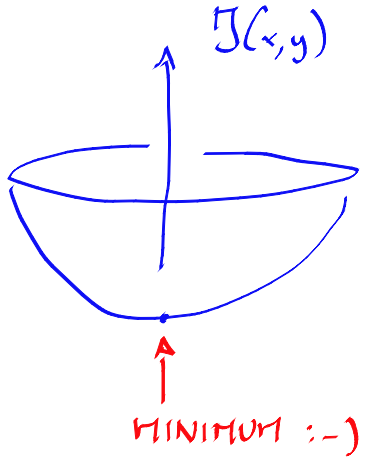
→ **Calcul de la taille requise pour la matrice active**

# Méthodes itératives

**Matrice définie positive**

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

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




Soit $\mathbf{x}^0$ une approximation initiale de la solution exacte $\mathbf{x}$ de (7.4),

il s'agit alors construire une série d'approximations $\mathbf{x}^i \to \mathbf{x}$
de la manière suivante

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{d}^k$$

AMPLITUDE DE LA DESCENTE

DIRECTION DE LA DESCENTE

# Méthode du gradient

(méthode de la plus grande pente)

$$\nabla J(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$$

$$\mathbf{d}^k = -\underbrace{(\mathbf{A}\mathbf{x}^k - \mathbf{b})}_{\mathbf{r}^k}$$

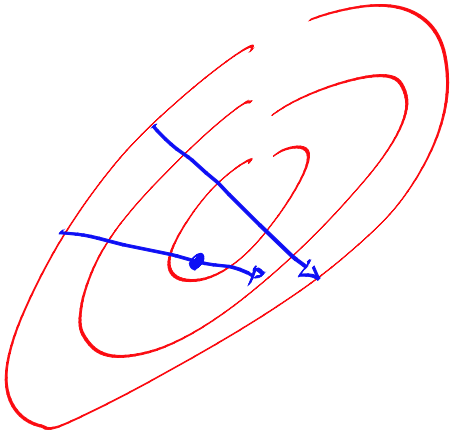
***Calcul de la plus grande pente***

Trouver $\alpha_k \geq 0$ tel que

$$J(\mathbf{x}^k + \alpha_k \mathbf{d}^k) = \min_{\alpha \geq 0} \underbrace{\left((\mathbf{x}^k + \alpha \mathbf{d}^k) \cdot \mathbf{A}(\mathbf{x}^k + \alpha \mathbf{d}^k) - \mathbf{b} \cdot (\mathbf{x}^k + \alpha \mathbf{d}^k)\right)}_{J((\mathbf{x}^k + \alpha \mathbf{d}^k))}$$

# Méthode du gradient

(méthode de la plus grande pente)



$$\frac{dJ(\mathbf{x}^k + \alpha_k \mathbf{d}^k)}{d\alpha} = 0$$

$$\downarrow$$

$$\left(\mathbf{A}(\mathbf{x}^k + \alpha_k \mathbf{d}^k) - \mathbf{b}\right) \cdot \mathbf{d}^k = 0$$

$$\downarrow$$

$$\underbrace{\left(\mathbf{A}\mathbf{x}^k - \mathbf{b}\right)}_{-\mathbf{d}^k} \cdot \mathbf{d}^k = -\alpha_k \mathbf{d}^k \cdot \mathbf{A}\mathbf{d}^k$$

$$\downarrow$$

$$\frac{\mathbf{r}_k \cdot \mathbf{r}^k}{\mathbf{r}^k \cdot \mathbf{A}\mathbf{r}^k} = \alpha_k$$

***Calcul du pas optimal***

# La méthode de la plus grande pente est une méthode lente !

**Analyse de la convergence**

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha\left(\mathbf{A}\mathbf{x}^k - \mathbf{b}\right)$$

$$\mathbf{x}^{k+1} - \mathbf{x} = \mathbf{x}^k - \mathbf{x} - \alpha\left(\mathbf{A}\mathbf{x}^k - \mathbf{b}\right)$$

$$\mathbf{x}^{k+1} - \mathbf{x} = \mathbf{x}^k - \mathbf{x} - \alpha\left(\mathbf{A}\mathbf{x}^k - \mathbf{b}\right) + \alpha\left(\mathbf{A}\mathbf{x} - \mathbf{b}\right)$$

$$\underbrace{\mathbf{x}^{k+1} - \mathbf{x}}_{\mathbf{e}^{k+1}} = (\boldsymbol{\delta} - \alpha\mathbf{A})\underbrace{(\mathbf{x}^k - \mathbf{x})}_{\mathbf{e}^k}$$

$$\|\mathbf{e}^{k+1}\| \leq \|\boldsymbol{\delta} - \alpha\mathbf{A}\|\,\|\mathbf{e}^k\|$$

$\mathbf{x}^k - \mathbf{x}$ = 0

$\mathbf{A}\mathbf{x} - \mathbf{b}$ = SOLUTION :-)

$\mathbf{e}^{k+1}$ : ERREUR ITERATION k+1

$\mathbf{e}^k$ : ERREUR ITERATION k

$(\boldsymbol{\delta} - \alpha\mathbf{A})$ : REGARDER LES VALEURS PROPRES !

# Un peu d'algèbre linéaire

$$|1 - \alpha\lambda_j| \leq 1$$

$$\alpha\lambda_{max} \leq 2$$

$$\alpha \leq \frac{2}{\lambda_{max}}$$

$$\|\mathbf{u}\| = \sqrt{\mathbf{u} \cdot \mathbf{u}},$$

$$\|\mathbf{A}\| = \max_{\mathbf{u} \in \mathbb{R}_0} \frac{\|\mathbf{A}\mathbf{u}\|}{\|\mathbf{u}\|}.$$

**Choix d'un pas acceptable** →

$$\|\boldsymbol{\delta} - \alpha\mathbf{A}\| = 1 - \frac{\lambda_{min}}{\lambda_{max}}$$

$$= 1 - \frac{1}{\kappa(\mathbf{A})}$$

# Et pratiquement alors ?

NBR ITERATIONS

$$\left(1 - \frac{1}{\kappa(\mathbf{A})}\right)^n \leq \epsilon$$

ERREUR REQUISE

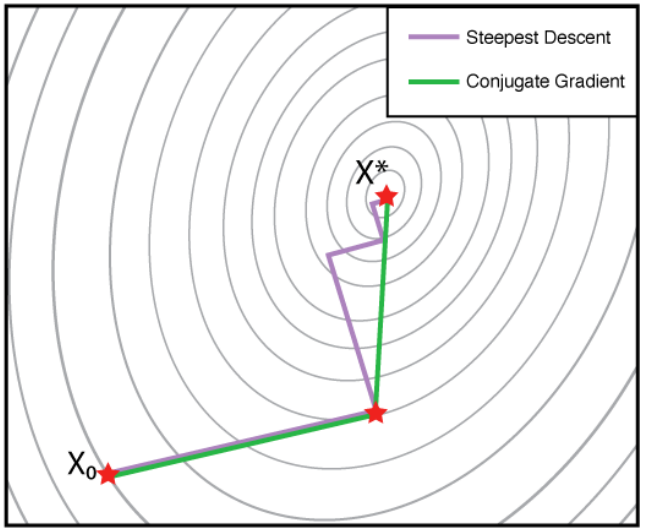
$$\log(\frac{1}{\epsilon}) \leq -n \underbrace{\log\left(1 - \frac{1}{\kappa(\mathbf{A})}\right)}_{\leq \frac{1}{\kappa(\mathbf{A})}}$$

$$\log(\frac{1}{\epsilon})\kappa(\mathbf{A}) \leq n$$

C'est très mauvais !

**Proportionnel à $O(h^{-2})$ pour une discrétisation d'un problème elliptique**

# Méthode des gradients conjugés

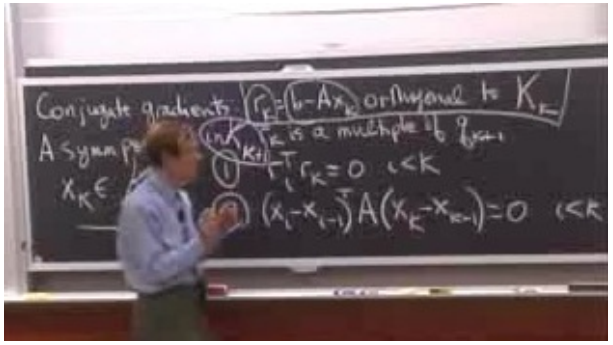



Le principe de l'algorithme est le suivant :

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k \qquad \text{avec } \alpha^k \text{ tel que } < \mathbf{r}^{k+1}\mathbf{r}^k > \; = \; 0$$

$$\mathbf{d}^{k+1} = \mathbf{r}^{k+1} + \beta^k \mathbf{d}^k \qquad \text{avec } \beta^k \text{ tel que } < \mathbf{d}^{k+1}\mathbf{A}\mathbf{d}^k > \; = \; 0$$

On peut montrer qu'un tel algorithme converge en (au plus) $n$ étapes pour une matrice définie positive.




$$\frac{1}{2}\log(\frac{2}{\epsilon})\sqrt{\kappa(\mathbf{A})} \quad \leq \quad n$$

**La convergence est nettement meilleure !**

# Implémentation des gradients conjugés

$$
\begin{aligned}
\alpha^k &= -\frac{< \mathbf{r}^k \mathbf{r}^k >}{< \mathbf{A}\mathbf{d}^k \mathbf{r}^k >} \\
\mathbf{x}^{k+1} &= \mathbf{x}^k + \alpha^k \mathbf{d}^k \\
\mathbf{r}^{k+1} &= \mathbf{r}^k + \alpha^k \mathbf{A}\mathbf{d}^k \\
\beta^k &= \frac{< \mathbf{r}^{k+1} \mathbf{r}^{k+1} >}{< \mathbf{r}^k \mathbf{r}^k >} \\
\mathbf{d}^{k+1} &= \mathbf{r}^{k+1} + \beta^k \mathbf{d}^k
\end{aligned}
$$

Attention, on n'effectue pas le produit matriciel $\mathbf{A}\mathbf{d}^k$, mais on assemble un vecteur $\mathbf{s}^k = \mathbf{A}\mathbf{d}^k$, tout comme on assemble un vecteur $\mathbf{r}^k = \mathbf{A}\mathbf{x}^k - \mathbf{b}$ ! Tout l'intérêt d'un algorithme itératif est de ne pas nécessiter le stockage d'une matrice de grande taille : c'est une implémentation *matrix-free* qu'on veut réaliser :-)

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

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

# Préconditionnement salut, les copains !

$$\begin{aligned} J(\mathbf{x}) &= \tfrac{1}{2}\mathbf{x} \cdot \mathbf{A}\mathbf{x} - \mathbf{b} \cdot \mathbf{x} \\ &= \tfrac{1}{2}(\mathbf{E}^{-1}\mathbf{y}) \cdot \mathbf{A}(\mathbf{E}^{-1}\mathbf{y}) - \mathbf{b} \cdot (\mathbf{E}^{-1}\mathbf{y}) \\ &= \tfrac{1}{2}\mathbf{y} \cdot \underbrace{(\mathbf{E}^{-T}\mathbf{A}\mathbf{E}^{-1})}_{\widetilde{\mathbf{A}}}\mathbf{y} - \underbrace{(\mathbf{E}^{-T}\mathbf{b})}_{\widetilde{\mathbf{b}}} \cdot \mathbf{y} \\ &= \widetilde{J}(\mathbf{y}) \end{aligned}$$

$y = Ex$

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

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

# Ah bon, et à quoi cela sert-il ?

***Le nouveau problème serait plus gentil…***
***(meilleur conditionnement)***

$$\mathbf{y}^{k+1} = \mathbf{y}^k - \alpha(\widetilde{\mathbf{A}}\mathbf{y}^k - \widetilde{\mathbf{b}})$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha \underbrace{\mathbf{E}^{-1}\mathbf{E}^{-T}}_{\mathbf{C}^{-1}}(\mathbf{A}\mathbf{x}^k - \mathbf{b})$$

***…et donc, ce schéma convergerait plus vite !***

# Aahhh, et comment trouver C ?

## ou les propriétés du bon préconditionneur

- Tout d'abord, le nombre de condition de $\mathbf{E}^{-T}\mathbf{A}\mathbf{E}^{-1}$ doit être le plus proche de l'unité. En tous cas, il doit au moins être nettement plus modeste que le nombre de condition de $\mathbf{A}$.
- Résoudre un système $\mathbf{C}\mathbf{x} = \mathbf{b}$ doit être économique, par rapport au système original,
- Obtenir le préconditionneur doit être peu coûteux.

**C= A est le choix optimal**
**C = I est le pire choix**

**C= A est le pire choix**
**C = I est le choix optimal**

***C = factorisation incomplète de Cholesky, est un bon compromis entre les 2 propriétés souhaitées.***