

# Journée des droits des femmes



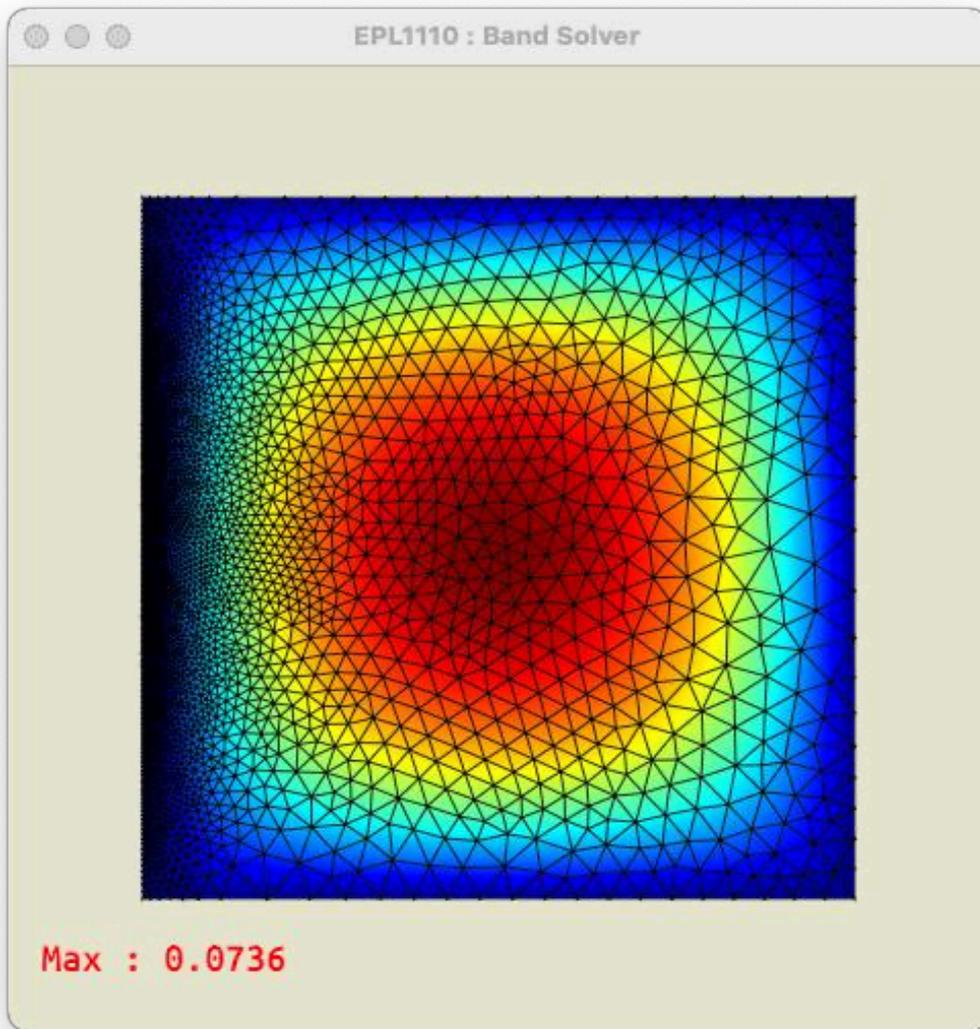
Margaret Heafield Hamilton, née le 17 août 1936, est une informaticienne et ingénieure système. Elle était directrice du département software engineering qui conçut le système embarqué du programme spatial Apollo

[https://www.youtube.com/watch?v=4sKY6\\_nBLG0](https://www.youtube.com/watch?v=4sKY6_nBLG0)

# Journée des droits des femmes



Comment communiquer en écrivant  
des textes lisibles et inclusifs  
en n'invisibilisant pas les femmes  
et les personnes non-binaires...



Un petit mot  
sur la résolution  
de grands systèmes linéaires

Résoudre  
un système  
triangulaire  
est facile...

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

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

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

**Backward substitution**

*Triangulation*

# Triangularisation par élimination de Gauss

$$\begin{array}{l}
 L1 \\
 L2
 \end{array}
 \begin{bmatrix}
 1 & 2 & 4 \\
 5 & 6 & 7 \\
 8 & 9 & 10
 \end{bmatrix}
 \begin{bmatrix}
 x \\
 y \\
 z
 \end{bmatrix}
 =
 \begin{bmatrix}
 1 \\
 2 \\
 3
 \end{bmatrix}$$

$$L2 = L2 - 5 L1$$

$$\begin{bmatrix}
 1 & 2 & 4 \\
 \underbrace{5-5} & \underbrace{6-10} & \underbrace{7-20} \\
 0 & -4 & -13
 \end{bmatrix}$$



*Karl Friedrich Gauss (1777-1855)*

# Triangularisation par élimination de Gauss

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

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

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



Karl Friedrich Gauss (1777-1855)

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

# Le problème discret

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

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

Matrice définie positive

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

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

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

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

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

$$\mathbf{Ax} = \mathbf{b}$$

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

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

Utilisation des notations habituelles de l'algèbre linéaire

# Deux grandes classes de méthodes de résolution

## Solveurs directs (Elimination gaussienne)

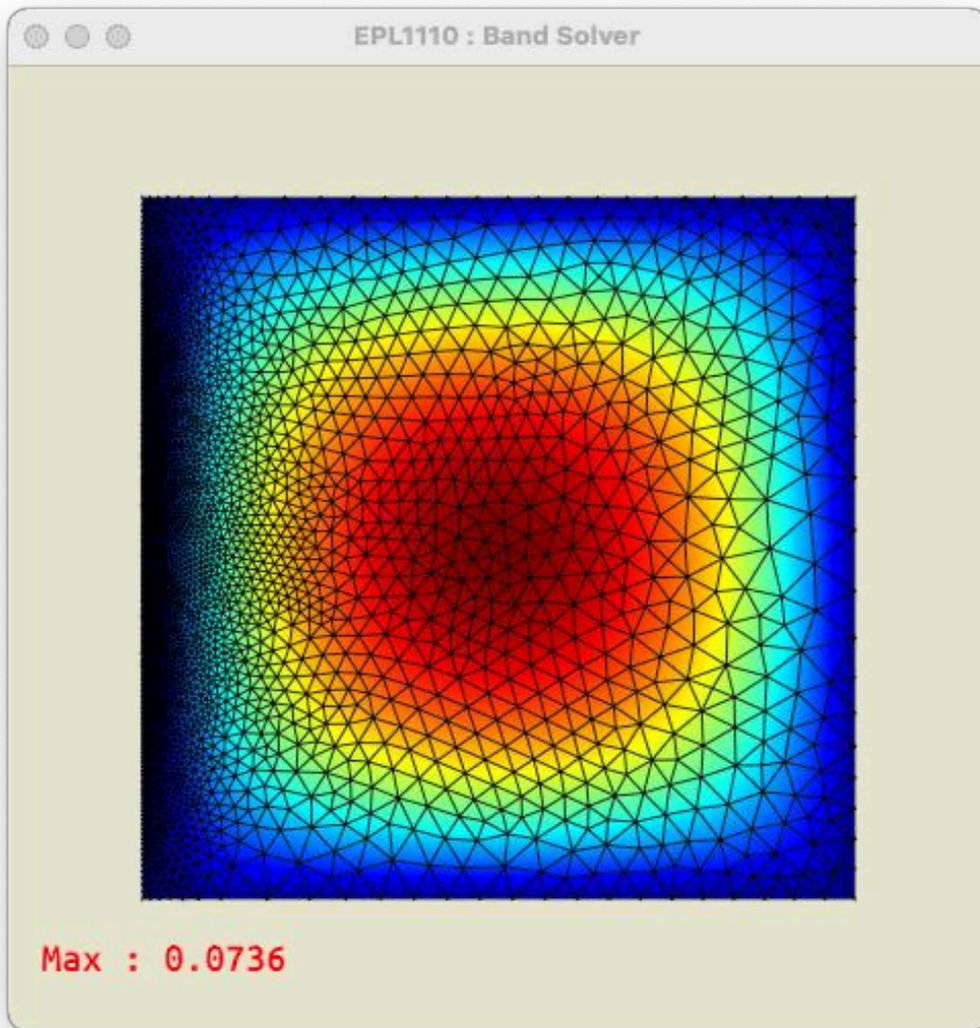
**Solveur de Gauss**  
**Solveur de Gauss bande**  
**Solveur de Gauss « skyline »**  
**Solveur de Gauss frontal**  
**Solveur de Gauss « Nested dissection »**

## Solveurs itératifs

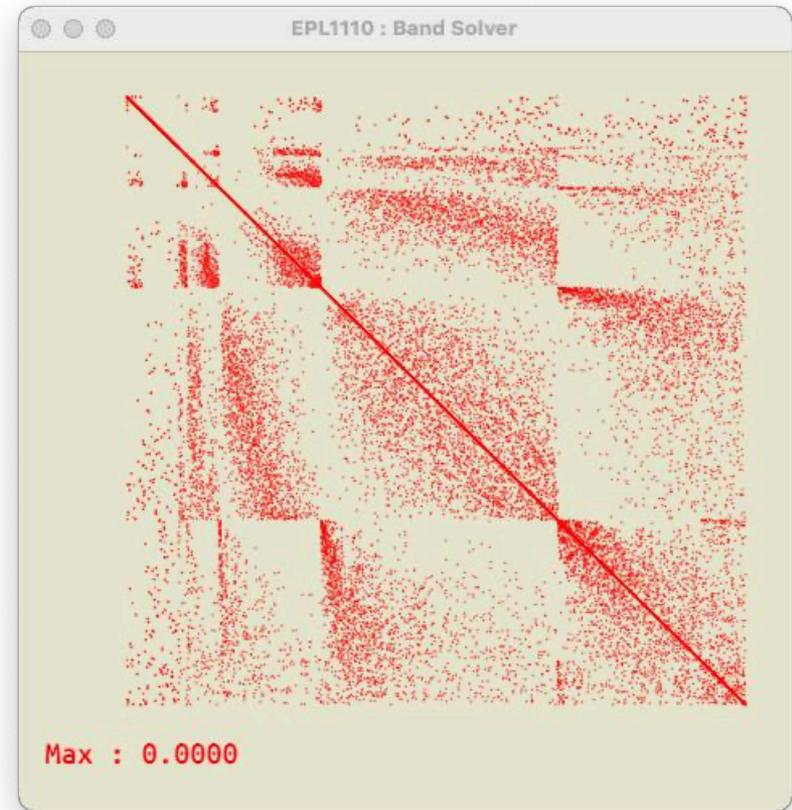
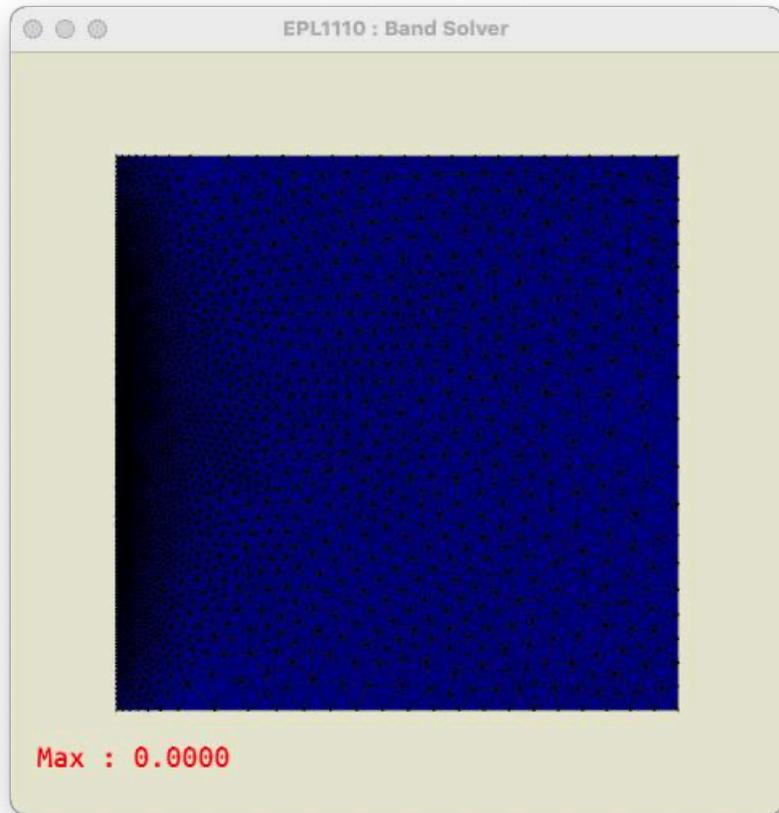
**Méthode de la plus grande pente**  
**Méthode des gradients conjugués**  
**Méthode de GMRES**

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

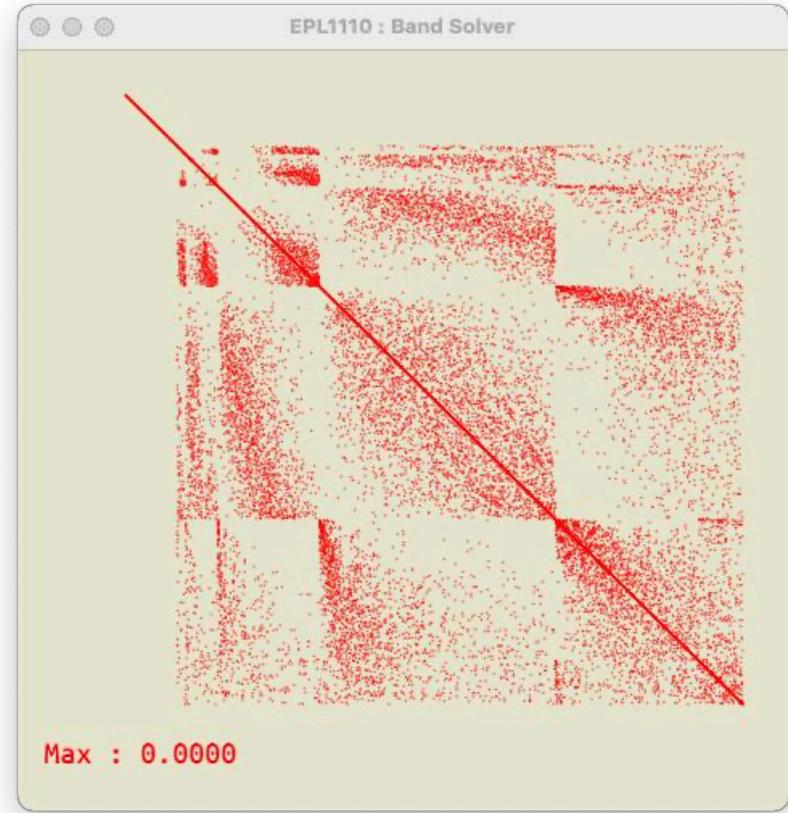
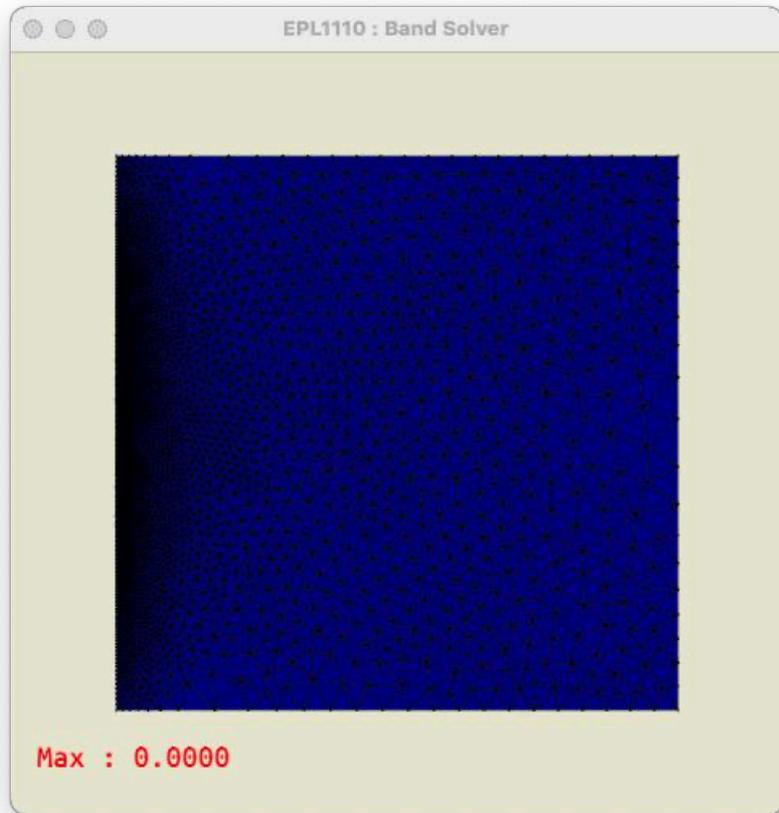




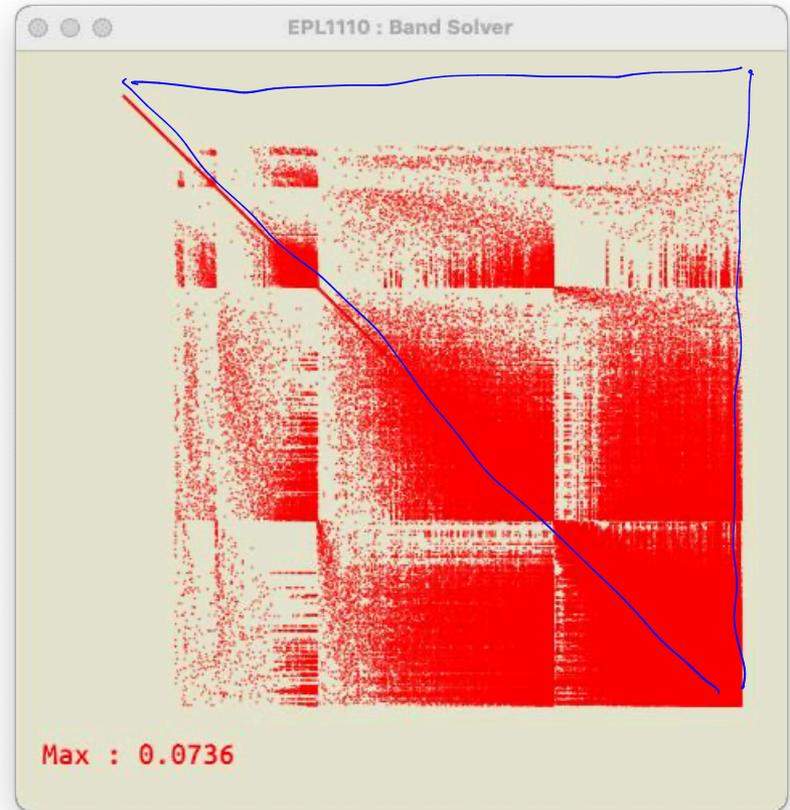
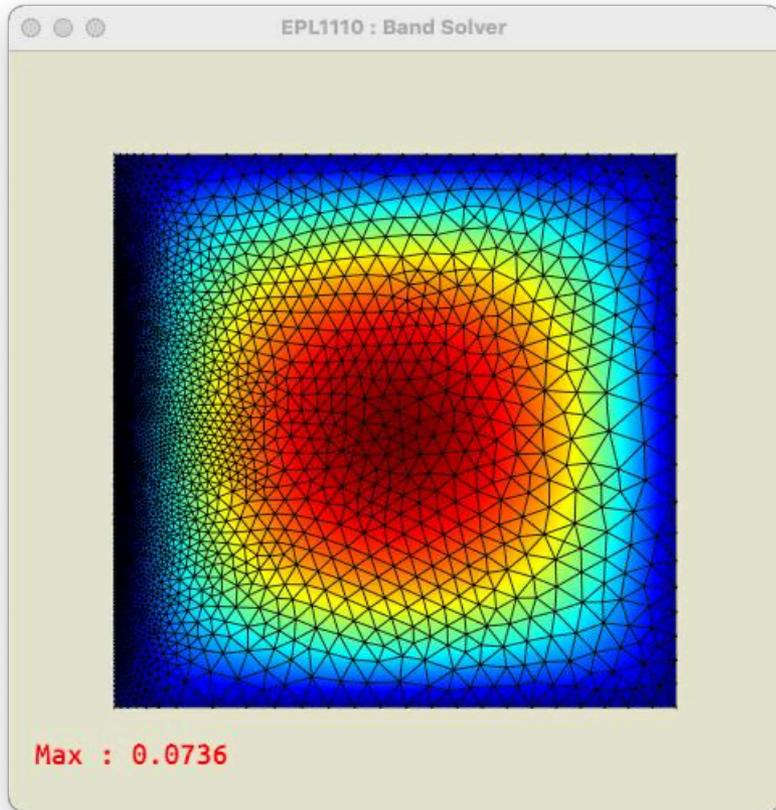
Revenons  
à notre  
problème:-)



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

$$L \underbrace{U}_{z} x = \underbrace{b}$$

- $Lz = b$

- $Ux = z$

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

$$l_{ii} = 1 \quad i = 1, \dots, n$$

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

$$b_{11} = \sqrt{a_{11}}$$

$$b_{i1} = \frac{a_{i1}}{b_{11}} \quad i = 2, \dots, n$$

$$b_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} b_{jk}^2} \quad j = 2, \dots, n$$

$$b_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} b_{ik}b_{jk}}{b_{jj}} \quad j = 2, \dots, n, i = j + 1, \dots, n$$

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

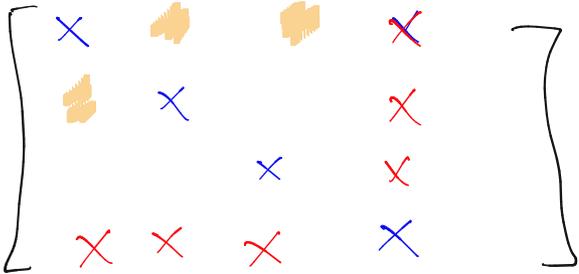
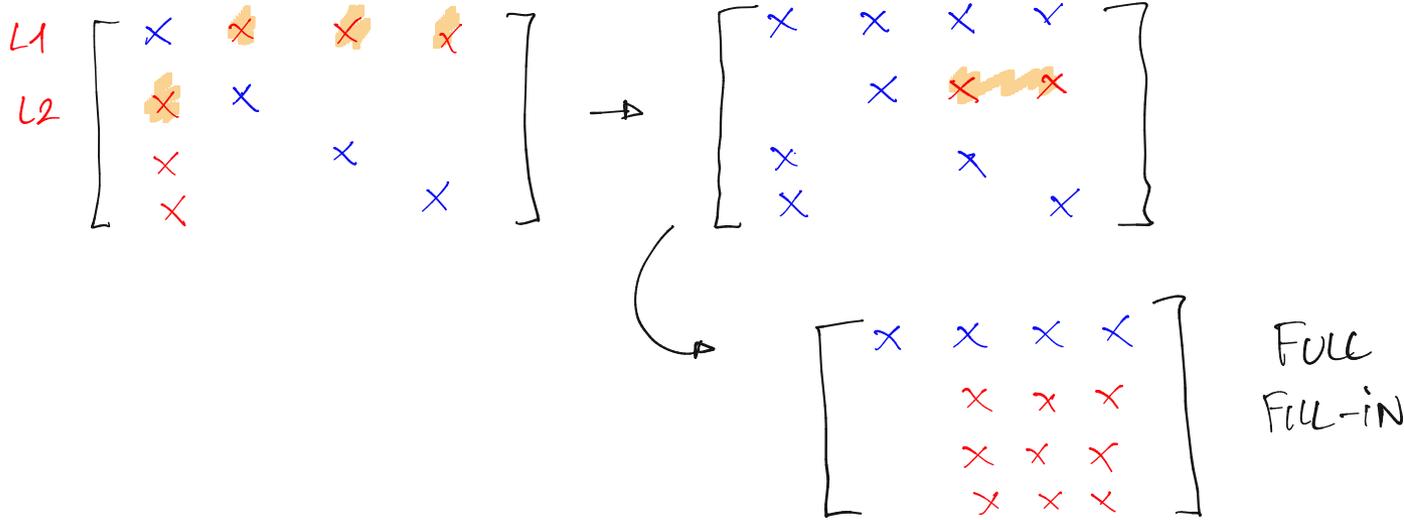
Plus intéressante que l'élimination gaussienne en termes d'opération,  
Mais valable que pour des matrices symétriques définies positives

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

$$\begin{array}{l}
 L1 \\
 L2
 \end{array}
 \mathbf{A} = \begin{bmatrix}
 \diamond & \diamond & \diamond & \diamond & \diamond & \diamond & \diamond \\
 \color{red}{\diamond} & \diamond & \color{red}{\times} & \color{red}{\times} & \color{red}{\times} & \color{red}{\times} & \color{red}{\times} \\
 \color{blue}{\diamond} & \color{blue}{\times} & \diamond & \color{blue}{\times} & \color{blue}{\times} & \color{blue}{\times} & \color{blue}{\times} \\
 \diamond & & & \diamond & & & \\
 \diamond & & & & \diamond & & \\
 \diamond & & & & & \diamond & \\
 \diamond & & & & & & \diamond
 \end{bmatrix}
 \quad
 \mathbf{U} = \begin{bmatrix}
 \diamond & & & & & & \\
 & \diamond & & & & & \\
 & & \diamond & & & & \\
 & & & \diamond & & & \\
 & & & & \diamond & & \\
 & & & & & \diamond & \\
 & & & & & & \diamond
 \end{bmatrix}$$

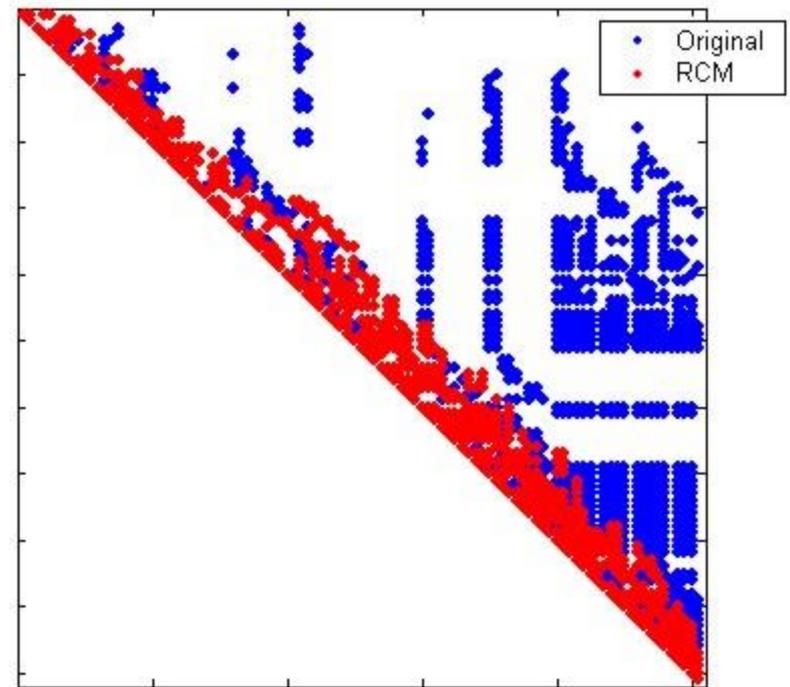
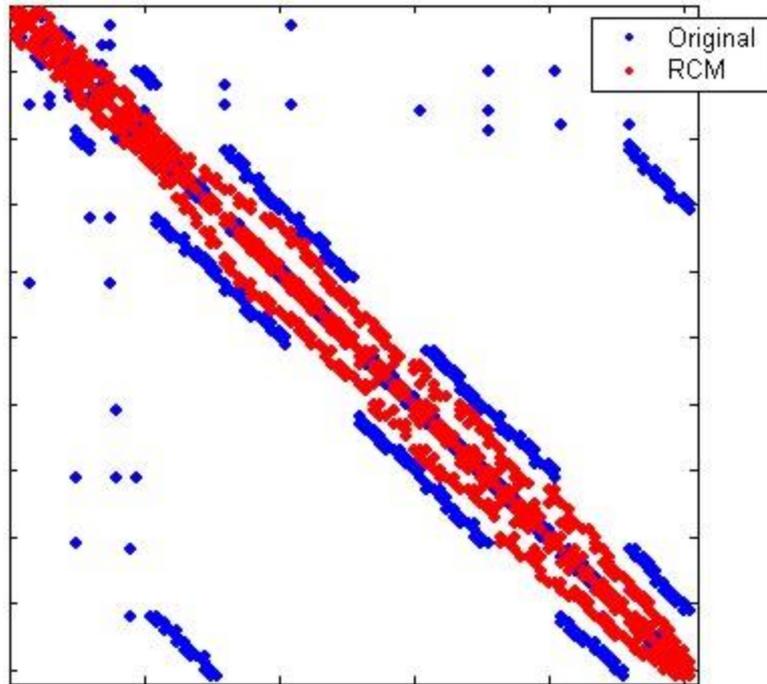
$$\mathbf{A} = \begin{bmatrix}
 \diamond & & & & & & \\
 & \diamond & & & & & \\
 & & \diamond & & & & \\
 & & & \diamond & & & \\
 & & & & \diamond & & \\
 & & & & & \diamond & \\
 & & & & & & \diamond \\
 \diamond & \diamond & \diamond & \diamond & \diamond & \diamond & \diamond
 \end{bmatrix}
 \quad
 \mathbf{U} = \begin{bmatrix}
 \diamond & & & & & & \\
 & \diamond & & & & & \\
 & & \diamond & & & & \\
 & & & \diamond & & & \\
 & & & & \diamond & & \\
 & & & & & \diamond & \\
 & & & & & & \diamond \\
 & & & & & & & \diamond
 \end{bmatrix}$$

# Fill-in

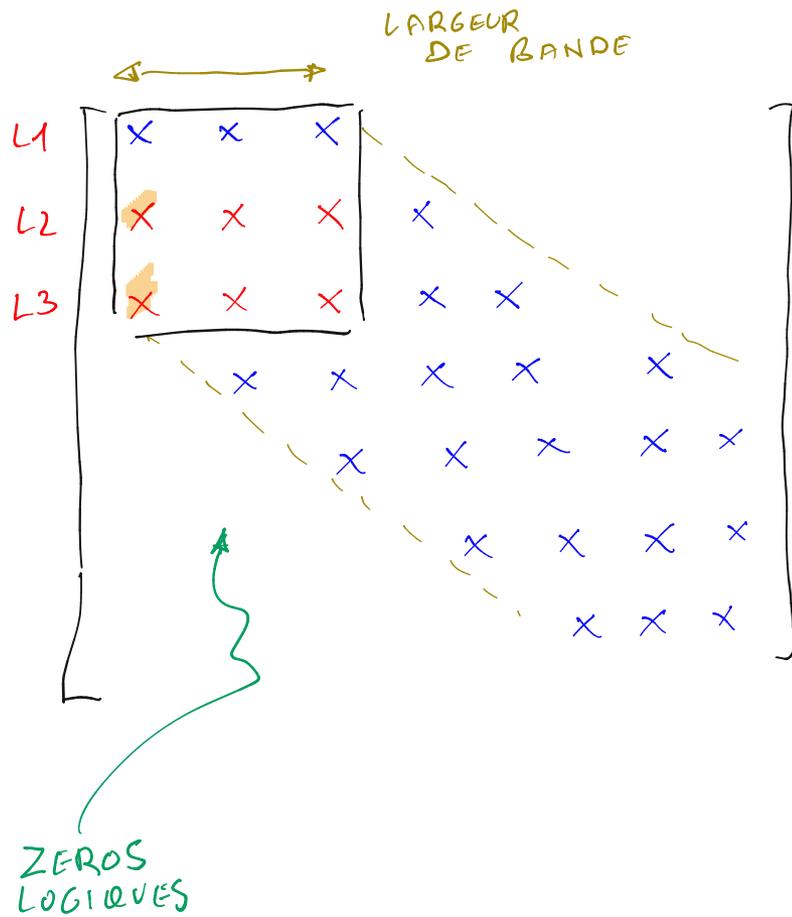


Seule, la numérotation des noeuds est cruciale !

# *Fill-in* et renumérotation des variables !

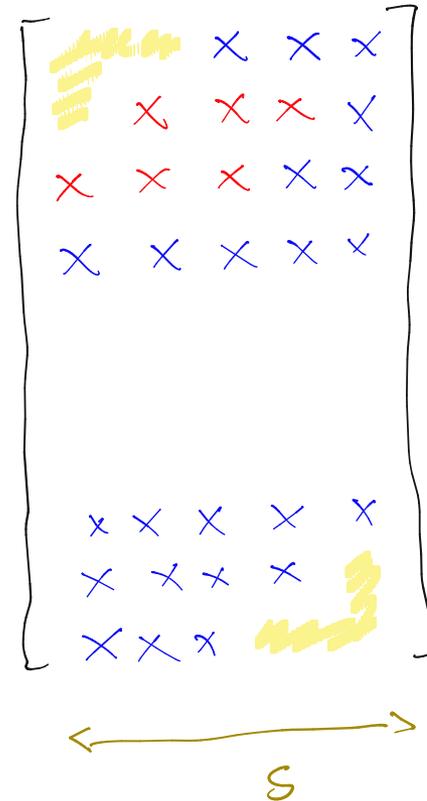


*Yannakakis, Computing the minimum fill-in is NP-complete, SIAM J. Algebraic and Discrete Methods, Vol. 2, 1981, 77-79.*



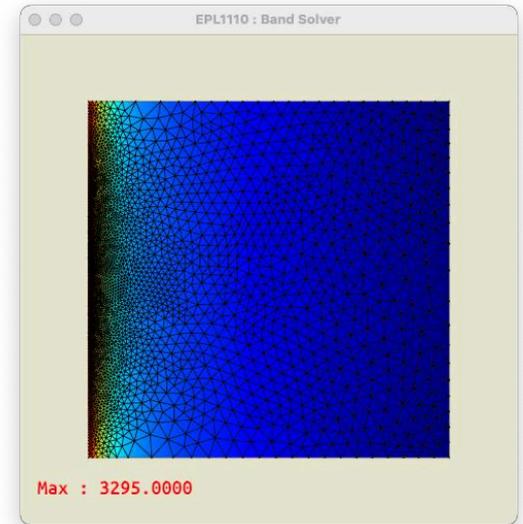
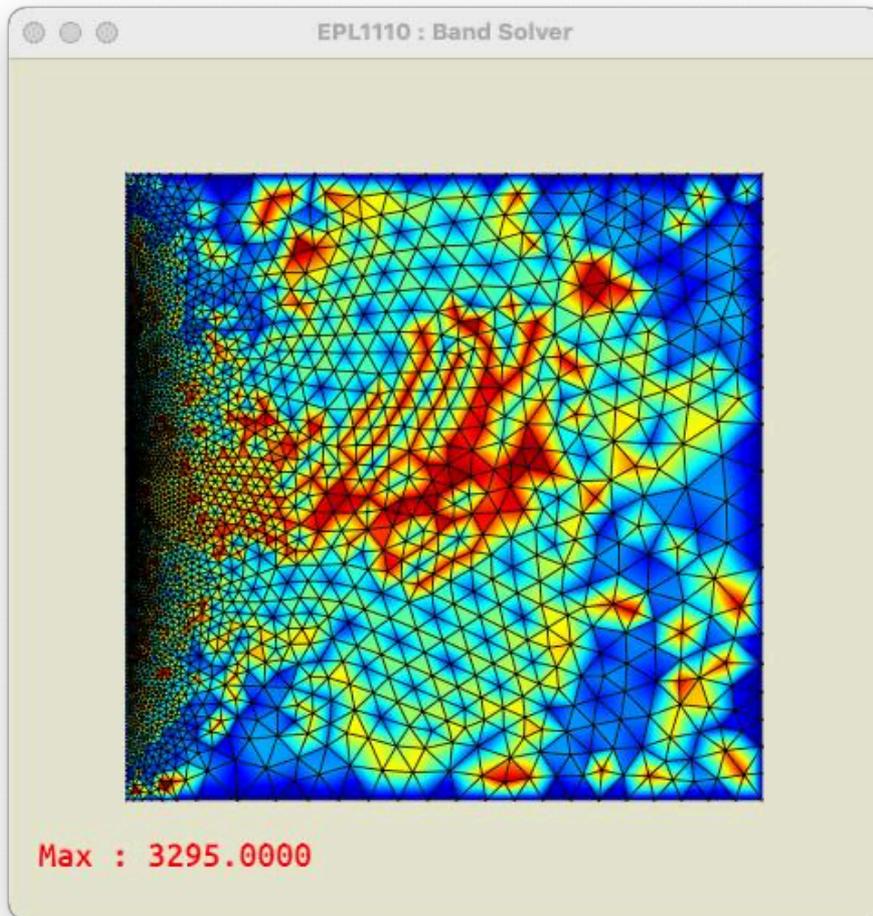
STOCKAGE  $\mathcal{O}(m\beta)$

COUT  $\mathcal{O}(m\beta^2)$



**Stockage  
de la matrice bande**

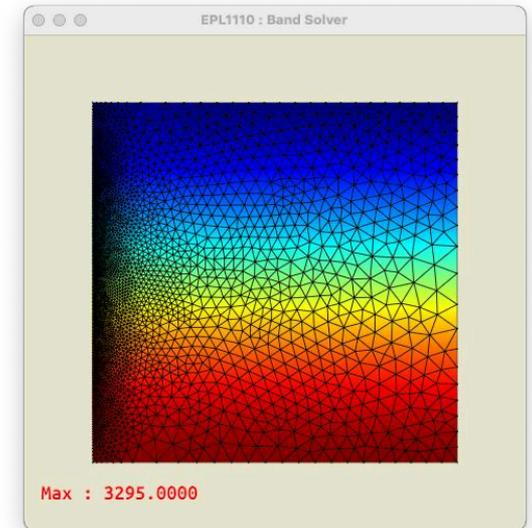
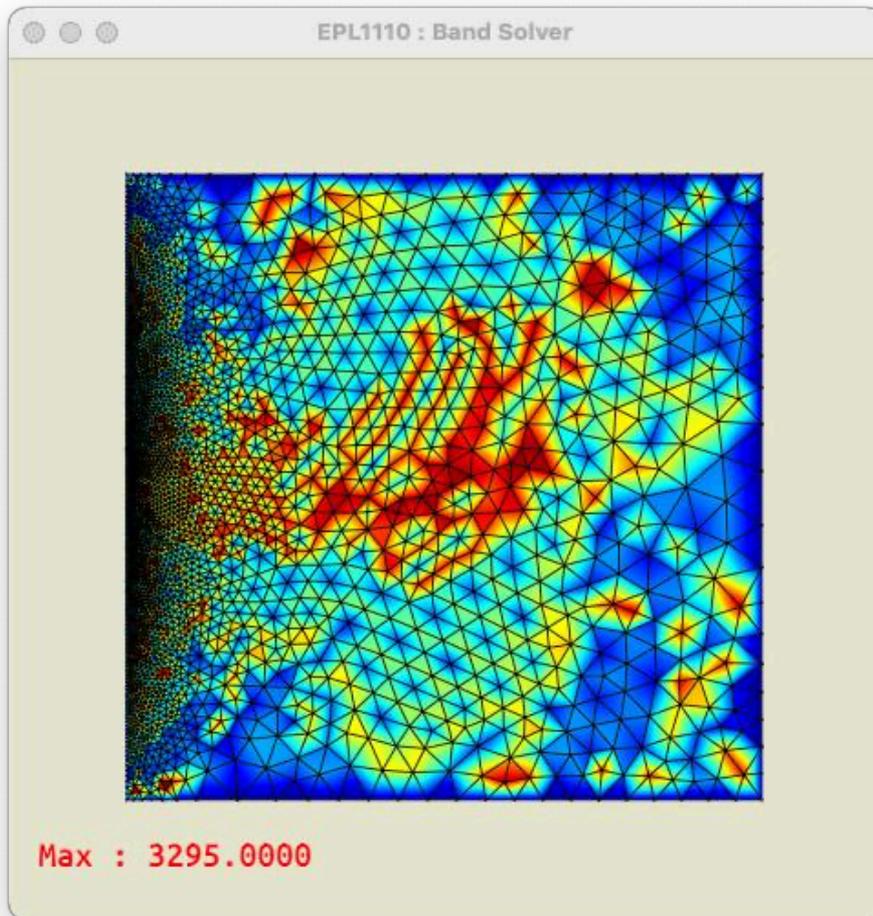
# En modifiant la numérotation des noeuds...



```
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...

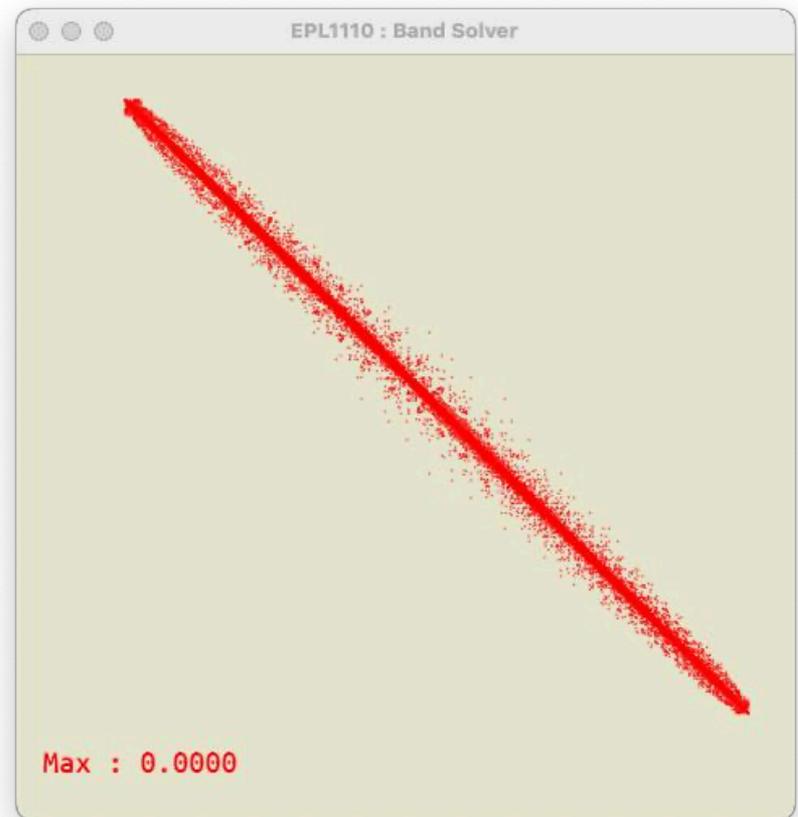
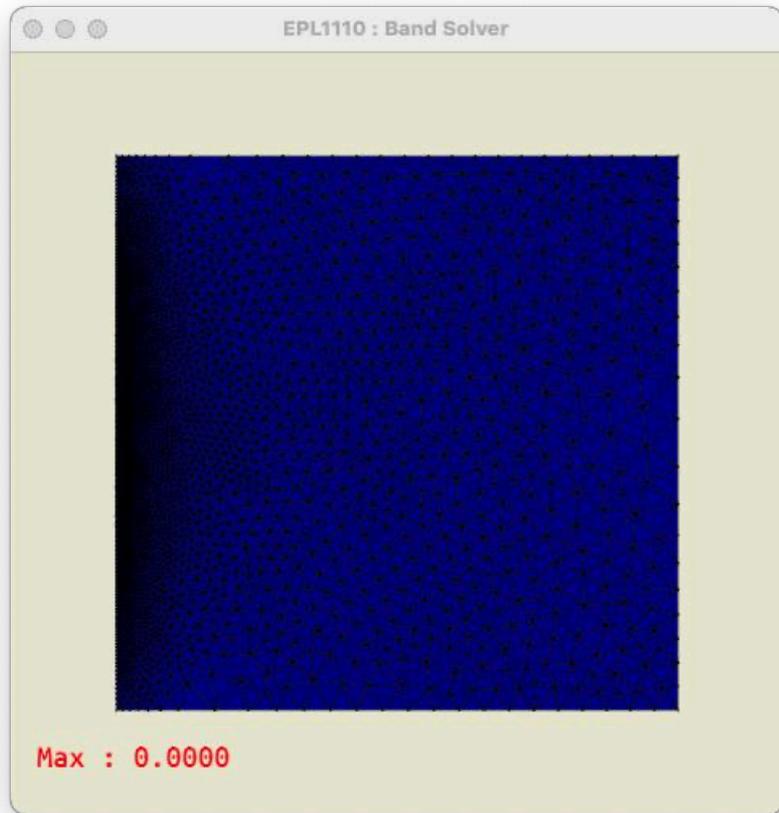


## BandSize Gaussian

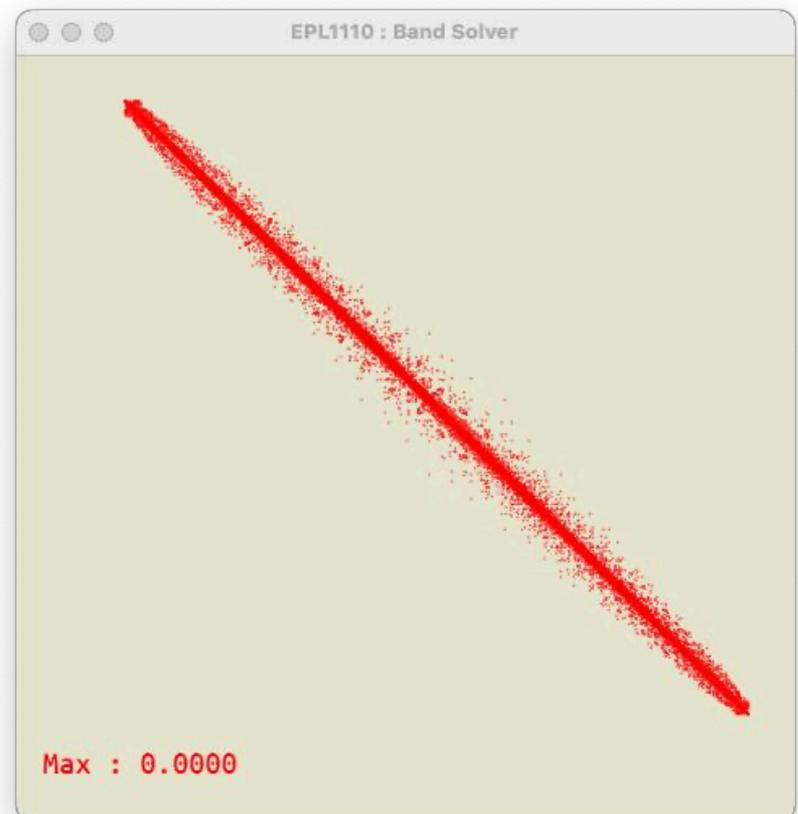
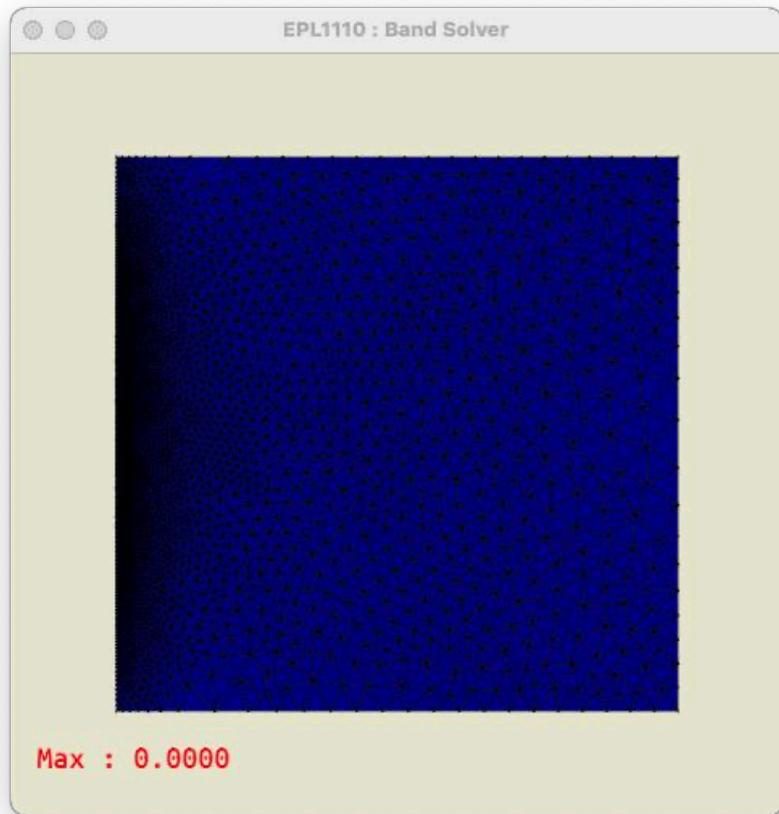
Matrix size : 3296  
Matrix band : 402  
Bytes required : 10626304

## Full Gaussian elimination

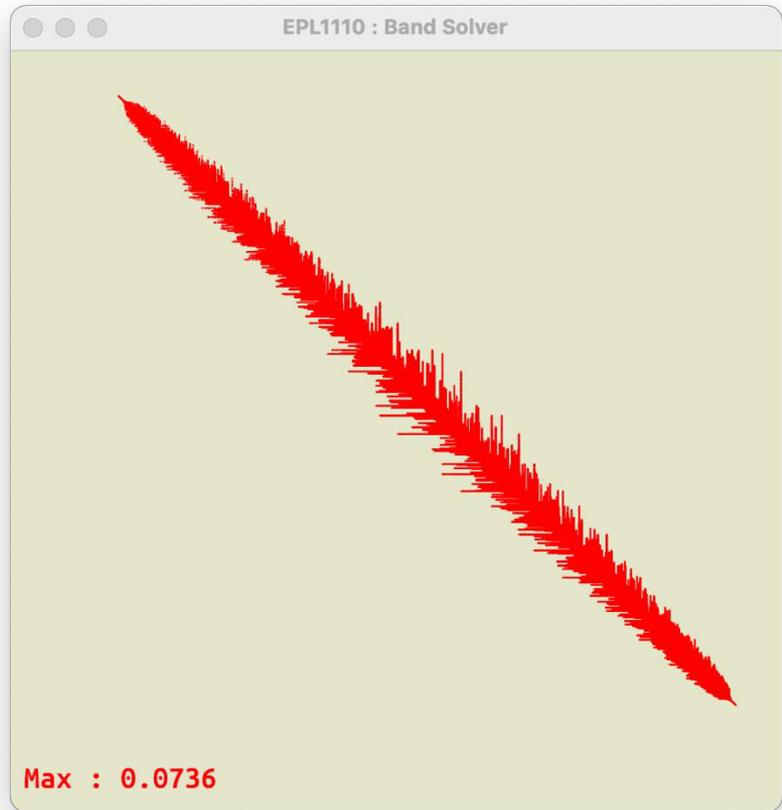
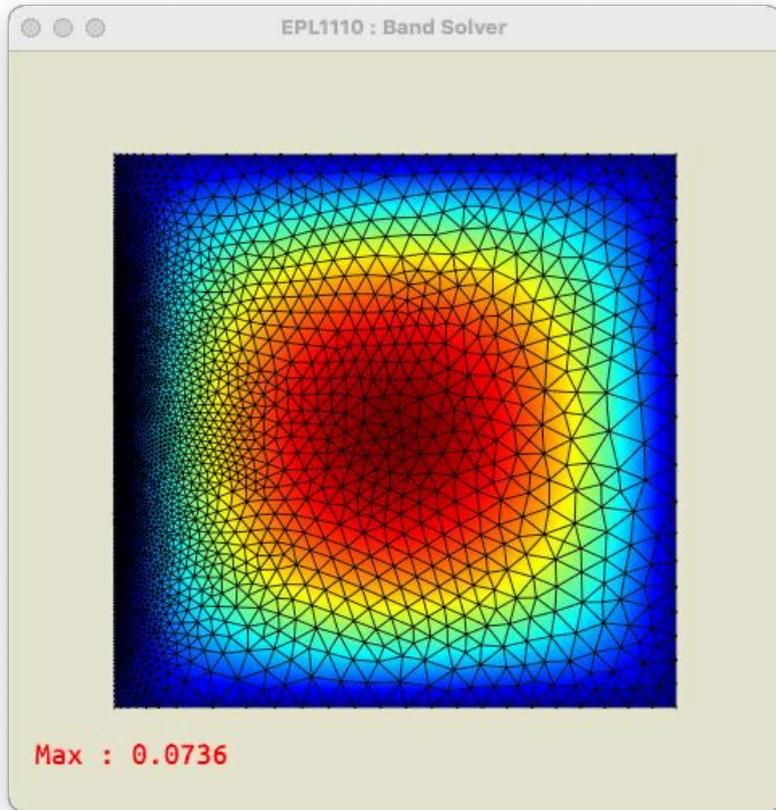
Matrix size : 3296  
Bytes required : 86935296



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

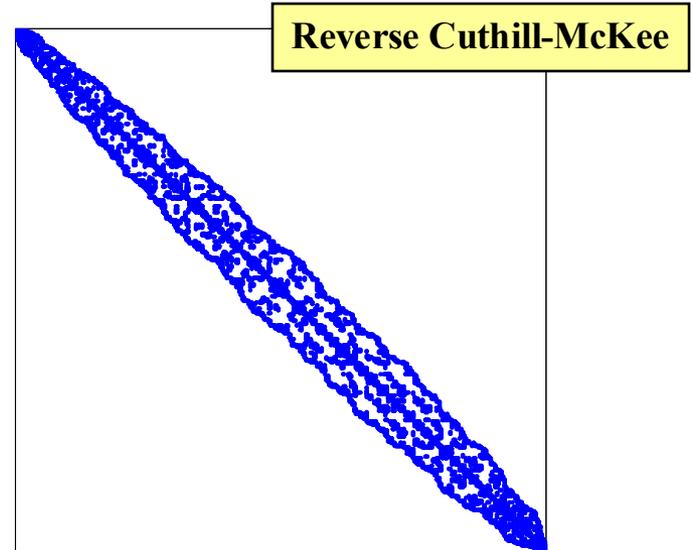
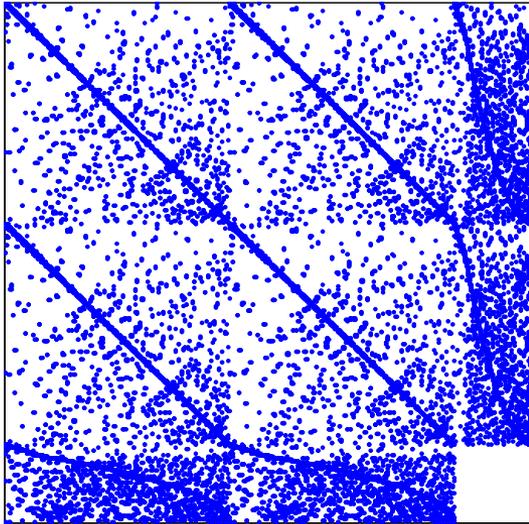


## Temps de calcul « cpu »

Heuristiques de rénumérotation

Utilisation des outils BLAS et LAPACK

# Heuristiques de renumérotation



Mimimum Degre Ordering

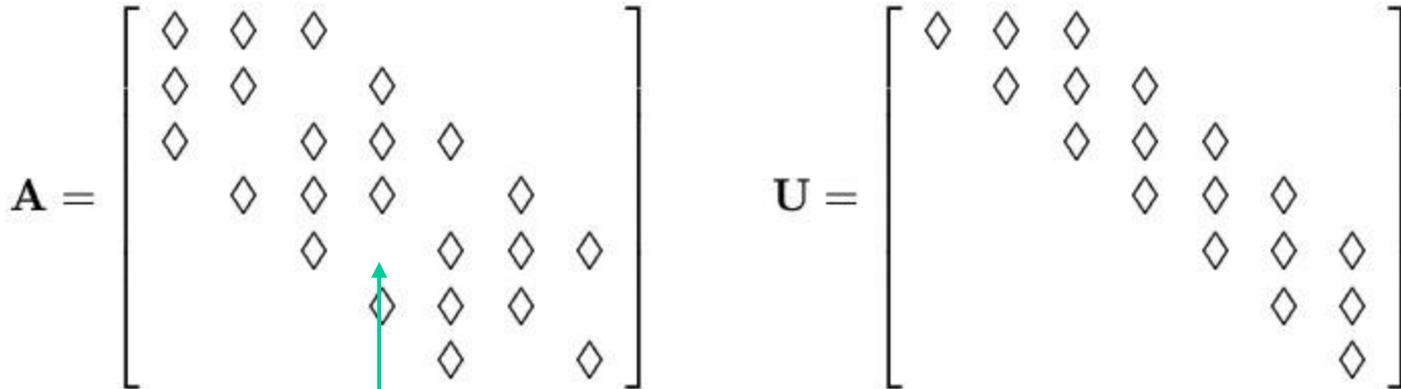


# Solveur de Gauss pour matrices bandes

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

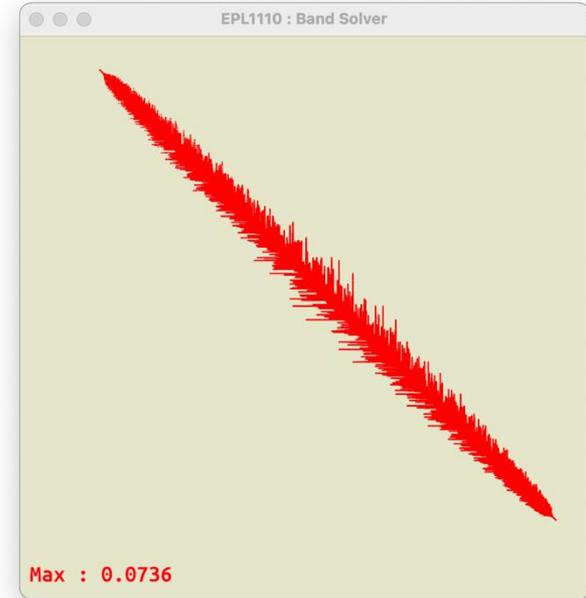
Coût calcul :  $O(n\beta^2)$

Concept de largeur de bande



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 4 :  
un solveur creux bande !

# Le C est un langage de bas niveau : les pointeurs ;-(

```
int main(void)
{
    int a = 4;
    printf(" ====   a === %d \n", a);
    printf(" ====   &a === %d \n", &a);
    int *b = &a; // &&a do not exist : why ?
    printf(" ====   &&a === %d \n", &b);
    exit(0);
}
```

Adresse de la  
case mémoire

int	a	1606415436	4
*int	&a	1606415424	1606415436
**int	&&a	...	1606415424

Valeur

L'utilisation des pointeurs permet d'écrire des codes très efficaces et rapides...  
Par contre, programmer est une tâche plus délicate et fastidieuse.  
Mais, la rapidité d'un code est critique pour la simulation numérique (et le jeu !) :  
Le langage C (ou C++) est bon choix ici !

# Le C est un langage de bas niveau les pointeurs ;-(

```
int a = 0;
int *b = &a;
b[0] = 4;
printf("==== *b == %d \n", *b);
printf("==== b[0] == %d \n", b[0]);
printf("==== a == %d \n", a);
printf("==== b == %d \n", b);
```

```
int    a    *b    b[0]
*int  &a    b
```

Adresse de la  
case mémoire

1606415436	4
1606415424	1606415436

**L'utilisation des pointeurs permet d'écrire des codes très efficaces et rapides...  
Par contre, programmer est une tâche plus délicate et fastidieuse.  
Mais, la rapidité d'un code est critique pour la simulation numérique (et le jeu !) :**  
**Le langage C (ou C++) est bon choix ici !**

On peut écrire n'importe où dans la mémoire de l'ordinateur !  
Ouuuuuups : c'est pas joli

```
int a = 0;
int *b = &a;
b[0] = 4;
b[1] = 3;
printf("==== *b == %d \n", *b);
printf("==== b[0] == %d \n", b[0]);
printf("==== b[1] == %d \n", b[1]);
printf("==== a == %d \n", a);
printf("==== b == %d \n", b);
printf("==== &b[0] == %d \n", &b[0]);
printf("==== &b[1] == %d \n", &b[1]);
```

int		b[1]	1606415440	3
int	a	*b	1606415436	4
*int	&a	b	1606415424	1606415436

Adresse de la  
case mémoire

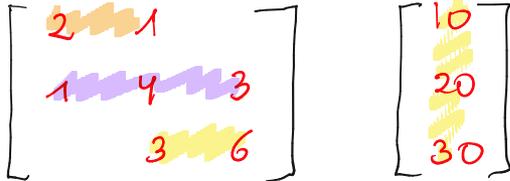
Et le pire, c'est que cela marche parfois...  
Parfois pas : **Segmentation fault**

# Et pour une matrice...

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

```
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;  
    femFullSystemInit(theSystem);  
  
    return theSystem;  
}
```

# Et zou...



```

double    B[0]          elem[0]  1606415390    10.0
double    B[1]          elem[1]  1606415394    20.0
double    B[2]          elem[2]  1606415398    30.0
double    A[0][0]       elem[3]  1606415402    2.0
double    A[0][1]       elem[4]  1606415406    1.0
double    A[0][2]       elem[5]  1606415410    0.0
double    A[1][0]       elem[6]  1606415414    1.0
double    A[1][1]       elem[7]  1606415418    4.0
double    A[1][2]       elem[8]  1606415422    3.0
double    A[2][0]       elem[9]  1606415426    0.0
double    A[2][1]       elem[10] 1606415430    3.0
double    A[2][2]       elem[11] 1606415434    6.0
**double  B             elem      1606415438    1606415390
**double  A[0]         elem+3    &elem[3]  1606415442    1606415402
**double  A[1]         elem+6    &elem[6]  1606415446    1606415414
**double  A[2]         elem+9    &elem[9]  1606415450    1606415426
**double  A             A        1606415454    1606415442

```

```

int i;
double *elem = malloc(sizeof(double) * size * (size+1));
mySystem->A = malloc(sizeof(double*) * size);
mySystem->B = elem;
mySystem->A[0] = elem + size;
mySystem->size = size;
for (i=1 ; i < size ; i++)
    mySystem->A[i] = mySystem->A[i-1] + size;

```

# Elimination gaussienne

```
double* femFullSystemEliminate(femFullSystem *mySystem)
{
    double **A, *B, factor;
    int i, j, k, size;
    A = mySystem->A;
    B = mySystem->B;
    size = mySystem->size;

    for (k=0; k < size; k++) {
        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; }

    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 (mySystem->B) ;
}
```

# Imposition d'une condition essentielle

```
void femFullSystemConstrain(femFullSystem *mySystem,
                            int myNode, double myValue)
{
    double **A, *B;
    int i, size;

    A = mySystem->A;
    B = mySystem->B;
    size = mySystem->size;

    for (i=0; i < size; i++) {
        B[i] -= myValue * A[i][myNode];
        A[i][myNode] = 0; }

    for (i=0; i < size; i++)
        A[myNode][i] = 0;

    A[myNode][myNode] = 1;
    B[myNode] = myValue;
}
```

# Et pour une matrice bande...

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

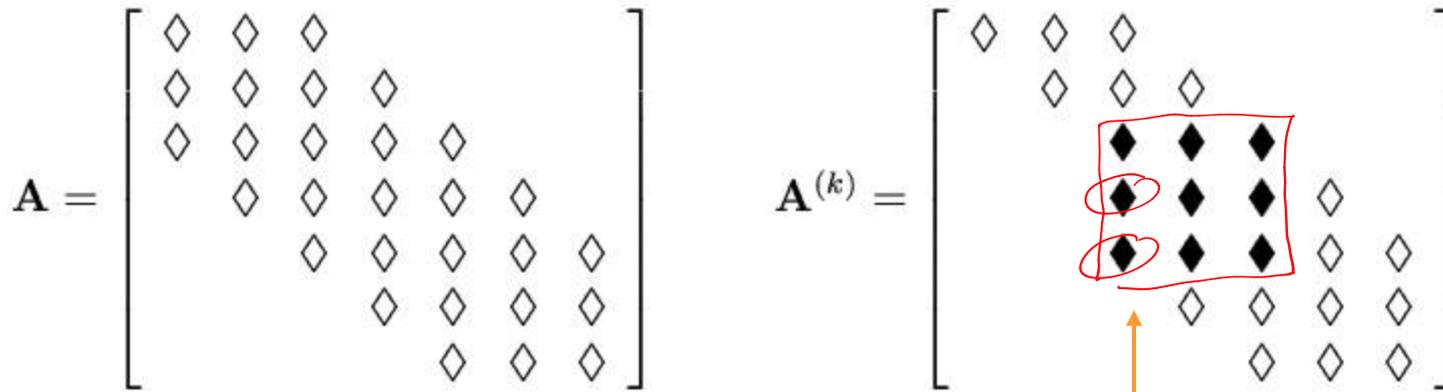
```
femBandSystem *femBandSystemCreate(int size, int band)  
{  
    femBandSystem *myBandSystem = malloc(sizeof(femBandSystem));  
    myBandSystem->B = malloc(sizeof(double)*size*(band+1));  
    myBandSystem->A = malloc(sizeof(double*)*size);  
    myBandSystem->size = size;  
    myBandSystem->band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

# Et zou..

double	B[0]	elem[0]	1606415390	10.0	
double	B[1]	elem[1]	1606415394	20.0	
double	B[2]	elem[2]	1606415398	30.0	
double	A[0][0]	elem[3]	1606415402	2.0	
double	A[1][0]=A[0][1]	elem[4]	1606415406	1.0	
double	A[2][0]=A[1][1]	elem[5]	1606415410	4.0	
double	A[2][1]=A[1][2]	elem[6]	1606415414	3.0	
double	A[2][2]	elem[7]	1606415418	6.0	
*double	B	elem	1606415438	1606415390	
*double	A[0]	elem+3	&elem[3]	1606415442	1606415402
*double	A[1]	elem+4	&elem[4]	1606415446	1606415406
*double	A[2]	elem+5	&elem[5]	1606415450	1606415410
**double	A		1606415454	1606415442	

```
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;
for (int i=1 ; i < size ; i++)
    myBandSystem->A[i] = myBandSystem->A[i-1] + band - 1;
```

# Solveur de Gauss frontal



Coût calcul :  $O(n\beta^2)$

Matrice active stockée sur la mémoire à accès rapide

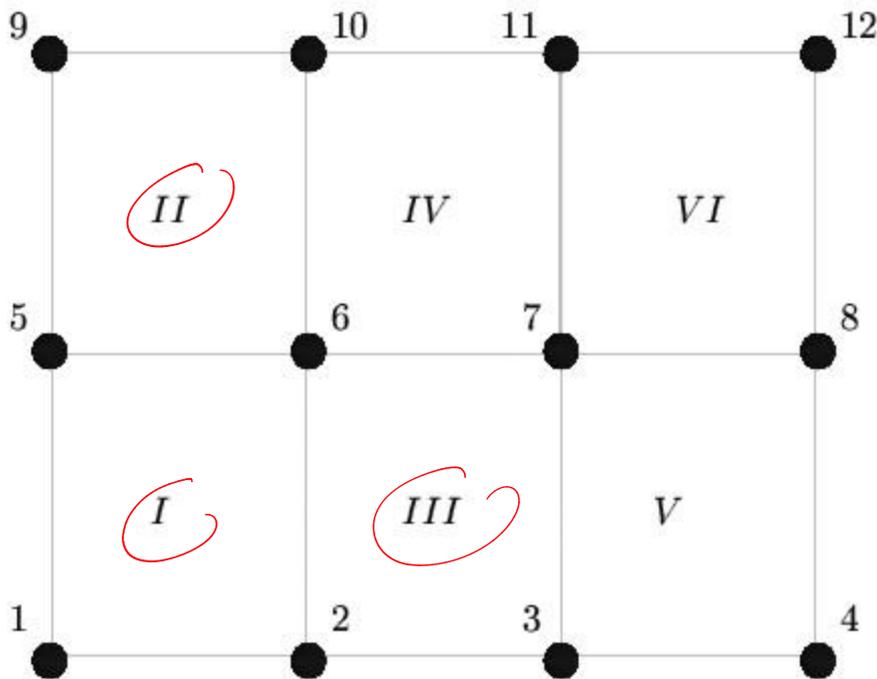
Le reste est stocké sur la mémoire secondaire

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**



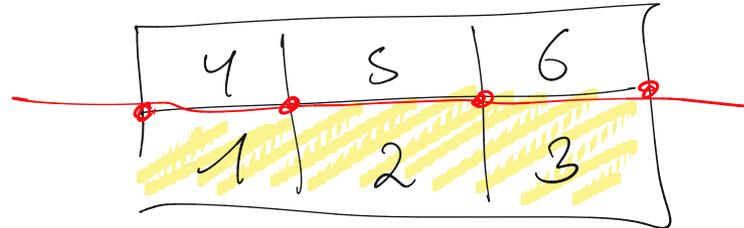
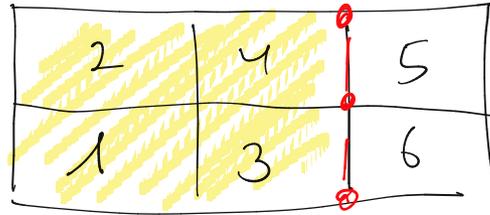
# Exemple

Seule, la numérotation des éléments est cruciale !



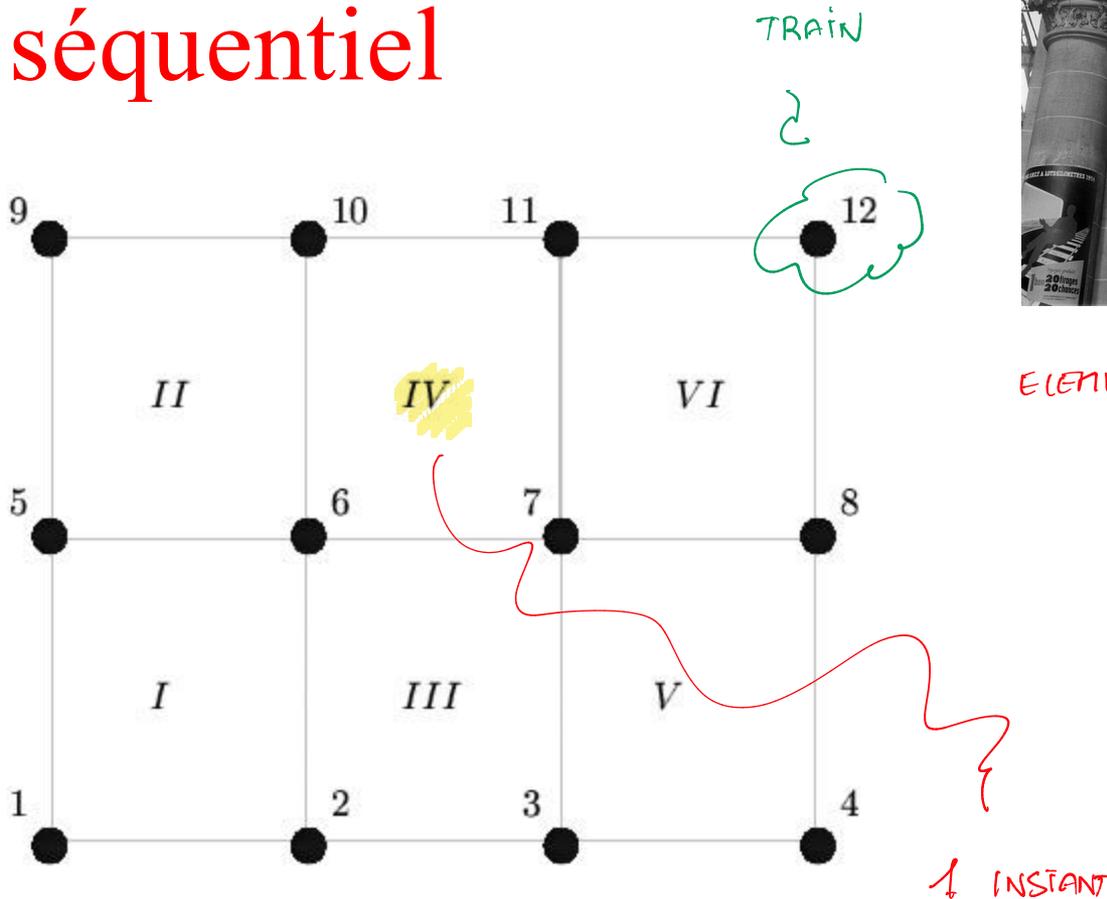
Elément	Sommets			
1	1	2	6	5
2	5	6	10	9
3	2	3	7	6
4	11	10	6	7
5	3	4	8	7
6	7	8	12	11

# Exemple



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

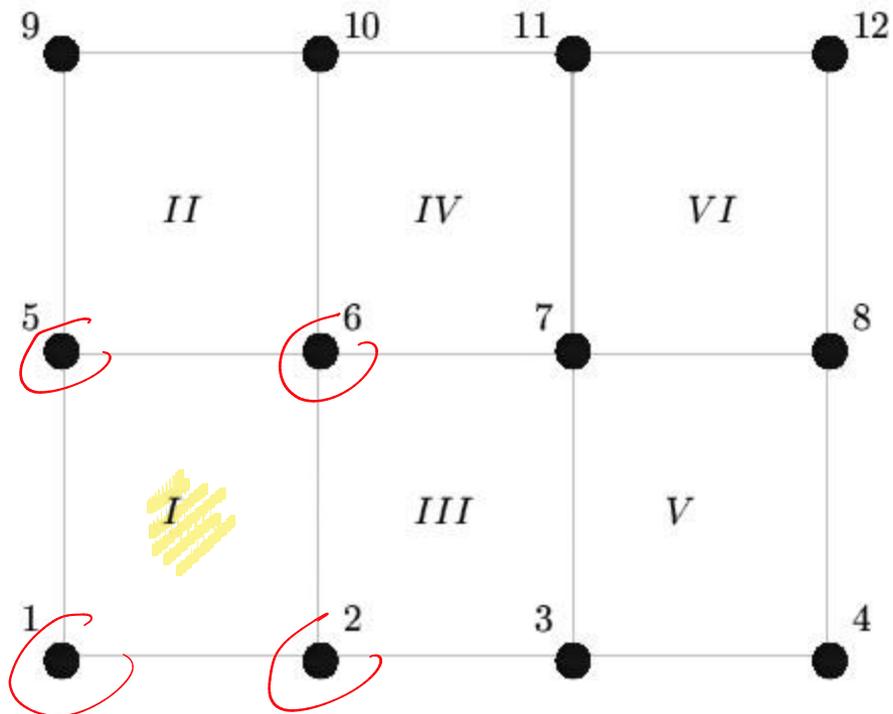
# L'assemblage est un processus séquentiel



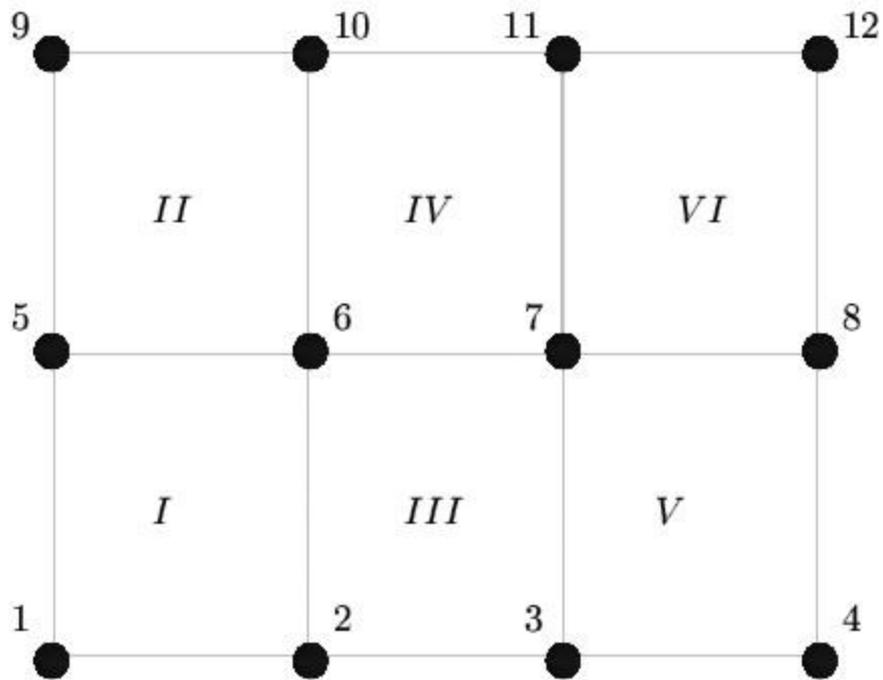
ELEMENT	1	12h	heure 1
	2	12h30	heure 2
	3	13h	heure 3
	4	13h30	heure 4
	5	14h	heure 5
	6	14h30	heure 6

1 INSTANT

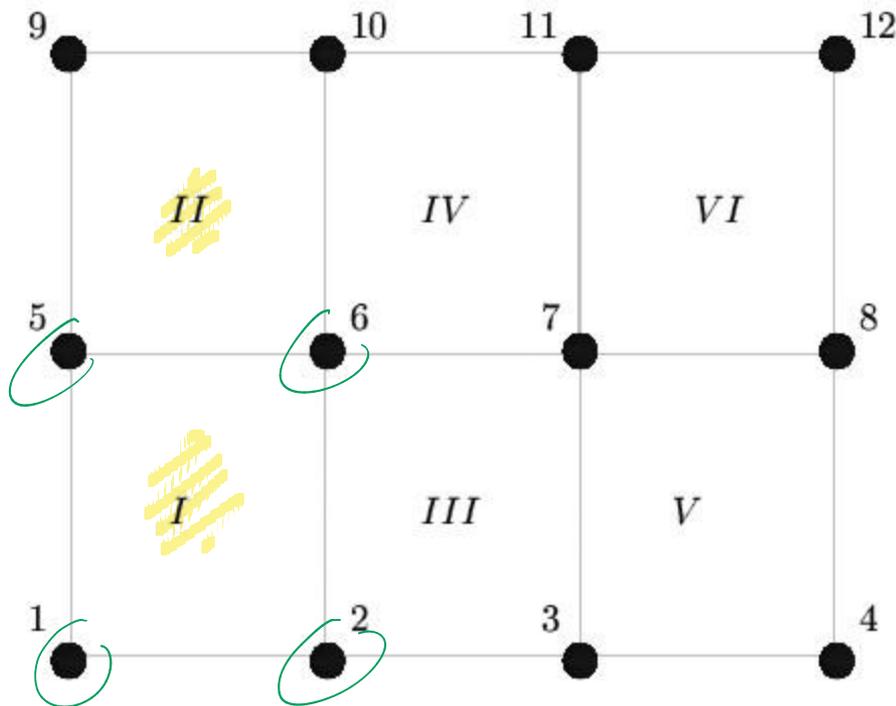
Chaque élément  
est une heure !



Chaque nœud  
est un train !



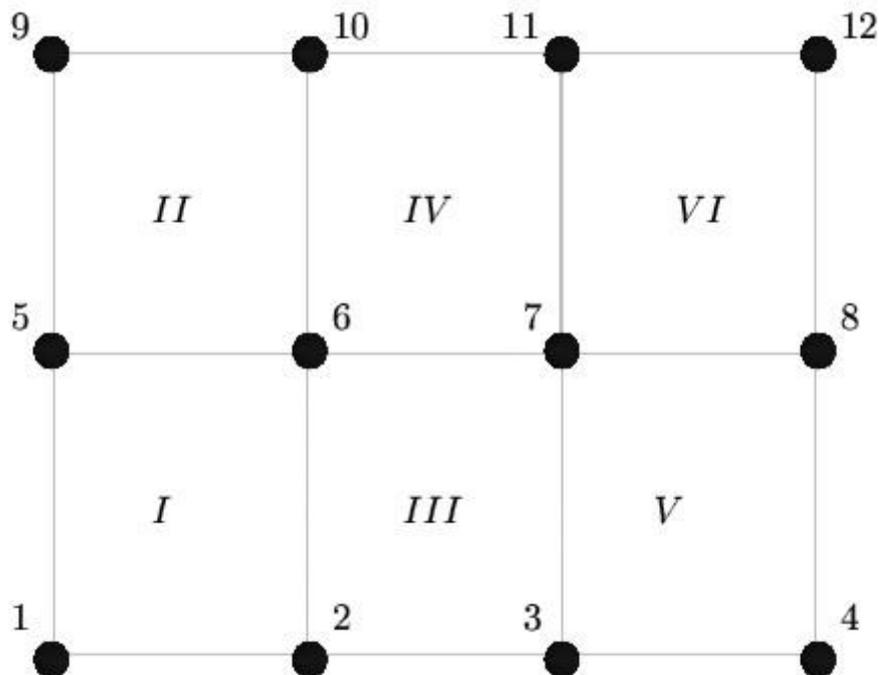
# Le train est passé !



ELEMENT

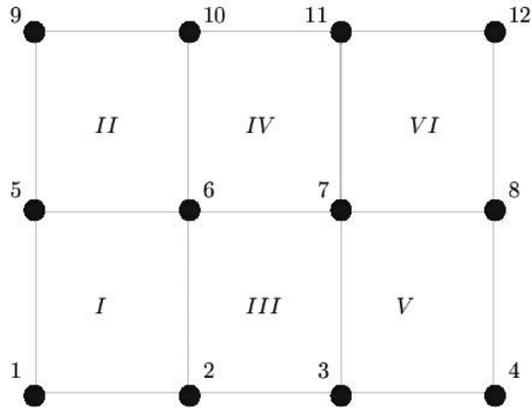
	1h	2h	3h	4h	5h	6h
1	[1]					
2	1		[3]			
3			3			[5 5]
4						
5	1	[2]				
6	1	2	3	[4]		
7			3	4	5	[6 6]
8						
9		[2]				
10		2				
11				[4]		
12				4		[6 6]

Heure ou élément  
où le train est passé !



Élément	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	<span style="border: 1px solid black; padding: 2px;">1</span>					
2	1		<span style="border: 1px solid black; padding: 2px;">3</span>			
3			3		<span style="border: 1px solid black; padding: 2px;">5</span>	
4					<span style="border: 1px solid black; padding: 2px;">5</span>	
5	1	<span style="border: 1px solid black; padding: 2px;">2</span>				
6	1	2	3	<span style="border: 1px solid black; padding: 2px;">4</span>		
7			3	4	5	<span style="border: 1px solid black; padding: 2px;">6</span>
8					5	<span style="border: 1px solid black; padding: 2px;">6</span>
9		<span style="border: 1px solid black; padding: 2px;">2</span>				
10		2		<span style="border: 1px solid black; padding: 2px;">4</span>		
11				4		<span style="border: 1px solid black; padding: 2px;">6</span>
12						<span style="border: 1px solid black; padding: 2px;">6</span>

# 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	



HEURES

	1	2	3	4	5	6
### 1	<span style="border: 1px solid red; padding: 2px;">1</span>	10	10	<span style="border: 1px solid blue; padding: 2px;">10</span>	8	<span style="border: 1px solid blue; padding: 2px;">8</span>
### 2	2	2	<span style="border: 1px solid blue; padding: 2px;">2</span>	11	11	11
### 3	6	6	6	<span style="border: 1px solid blue; padding: 2px;">6</span>	<span style="border: 1px solid blue; padding: 2px;">4</span>	<span style="border: 1px solid blue; padding: 2px;">12</span>
### 4	5	<span style="border: 1px solid blue; padding: 2px;">5</span>	3	3	<span style="border: 1px solid blue; padding: 2px;">3</span>	
### 5		<span style="border: 1px solid blue; padding: 2px;">9</span>	7	7	7	<span style="border: 1px solid blue; padding: 2px;">7</span>

Sommet	Elément de disparition					
1	<span style="border: 1px solid blue; padding: 2px;">1</span>					
2	1		<span style="border: 1px solid blue; padding: 2px;">3</span>			
3			3		<span style="border: 1px solid blue; padding: 2px;">5</span>	
4					<span style="border: 1px solid blue; padding: 2px;">5</span>	
5	1	<span style="border: 1px solid blue; padding: 2px;">2</span>				
6	1	2	3	<span style="border: 1px solid blue; padding: 2px;">4</span>		
7			3	4	5	<span style="border: 1px solid blue; padding: 2px;">6</span>
8					5	<span style="border: 1px solid blue; padding: 2px;">6</span>
9		<span style="border: 1px solid blue; padding: 2px;">2</span>				
10		2		<span style="border: 1px solid blue; padding: 2px;">4</span>		
11				4		<span style="border: 1px solid blue; padding: 2px;">6</span>
12						<span style="border: 1px solid blue; padding: 2px;">6</span>

1  
3  
5  
5  
2  
4  
4  
6  
6  
6  
2  
4  
6  
6

# Attribuer une voie libre à un train !

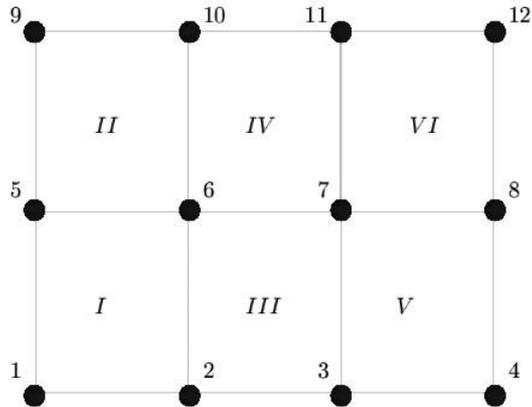


		Occupation de la matrice active lors de l'assemblage des éléments					
Elément		1	2	3	4	5	6
Destination							
1		1	10	10	10	4	12
2		2	2	2	11	11	11
3		6	6	6	6	8	8
4		5	5	3	3	3	
5			9	7	7	7	7
6							



**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		<span style="border: 1px solid black; padding: 2px;">1</span>	10	10	<span style="border: 1px solid black; padding: 2px;">10</span>	<span style="border: 1px solid black; padding: 2px;">4</span>	<span style="border: 1px solid black; padding: 2px;">12</span>
2		2	2	<span style="border: 1px solid black; padding: 2px;">2</span>	11	11	<span style="border: 1px solid black; padding: 2px;">11</span>
3		6	6	6	<span style="border: 1px solid black; padding: 2px;">6</span>	8	<span style="border: 1px solid black; padding: 2px;">8</span>
4		5	<span style="border: 1px solid black; padding: 2px;">5</span>	3	3	<span style="border: 1px solid black; padding: 2px;">3</span>	
5			<span style="border: 1px solid black; padding: 2px;">9</span>	7	7	7	<span style="border: 1px solid black; padding: 2px;">7</span>
6							

➔ Calcul de la taille requise pour la matrice active