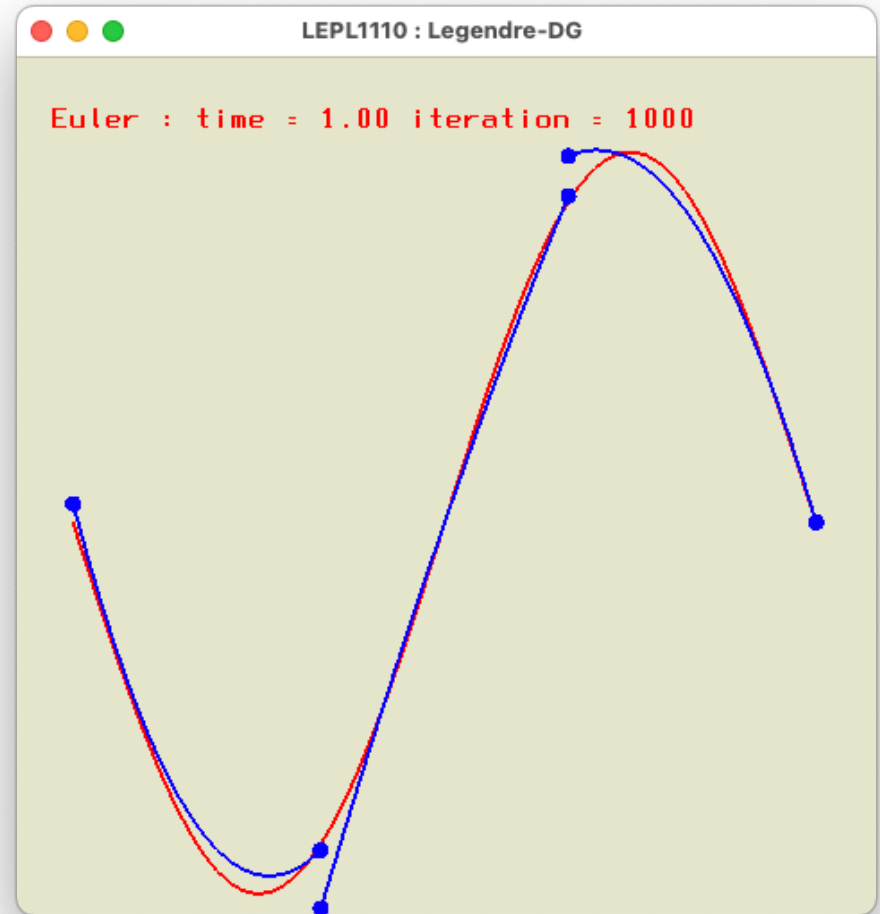


# Discontinuous Galerkin Method

$$\begin{cases} \frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} = 0 \\ u(t, L/2) = u(t, -L/2) \\ u(0, x) = \bar{u}(x) \end{cases}$$

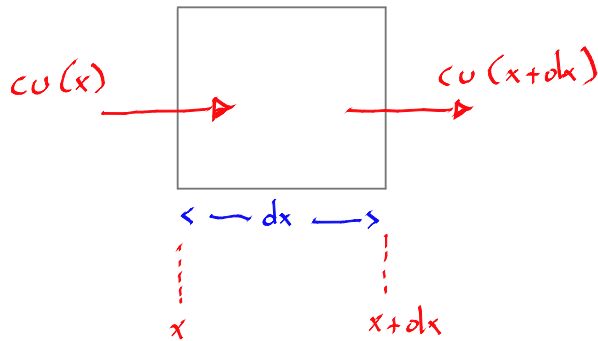


C'est quoi  
physiquement  
une équation  
de transport ?

$$\frac{\partial u}{\partial t} dx = \text{CE QUI ENTRE} - \text{CE QUI SORT}$$

$$\frac{\partial u}{\partial t} dx + \underbrace{cu|_{x+dx}}_{\text{CE QUI SORT}} - \underbrace{cu|_x}_{\text{CE QUI ENTRE}} = 0$$

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(cu) = 0$$



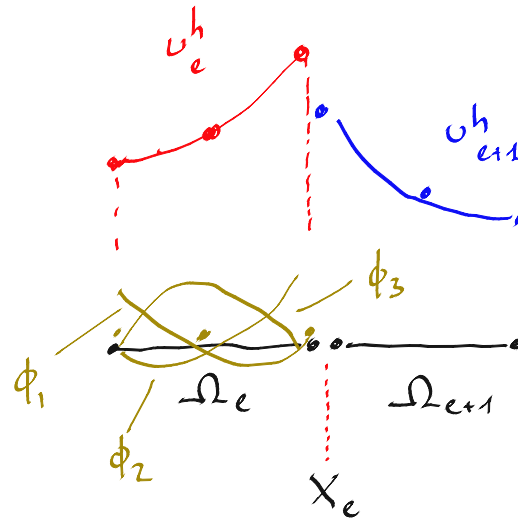
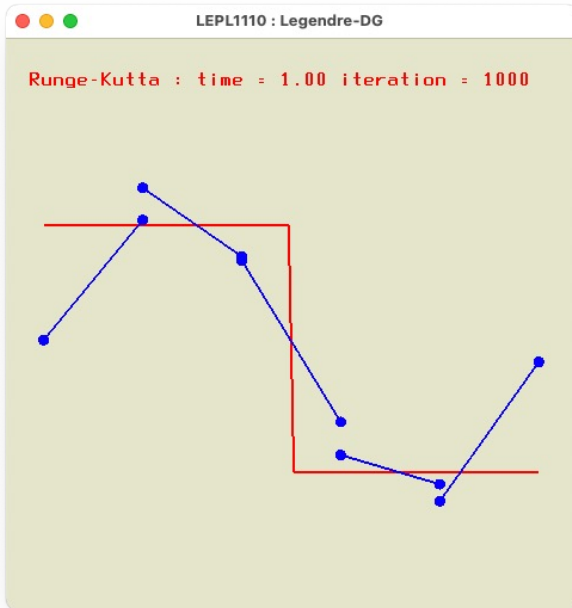
$$c f(x-ct) = u(x,t)$$

$\uparrow$  1 CST  
 $\uparrow$  1 CL

$$\frac{\partial u}{\partial t} = -cf'$$

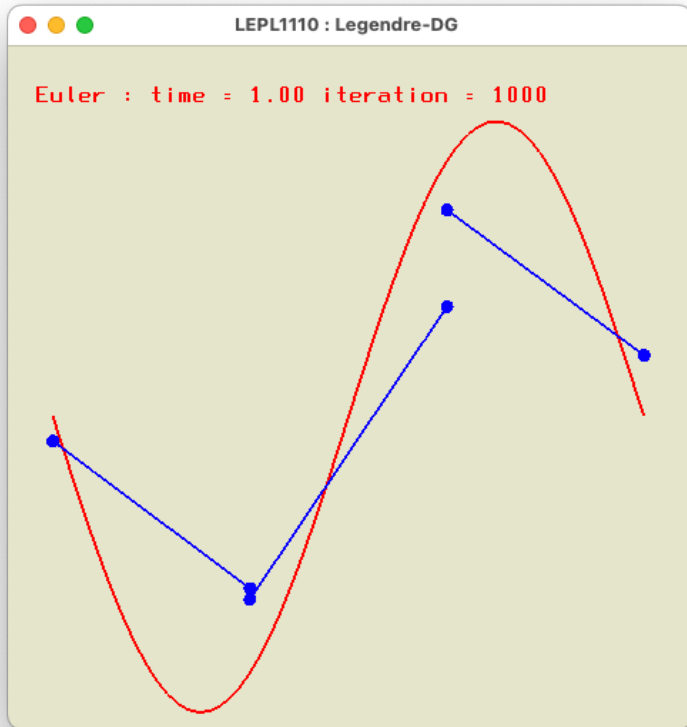
$$\frac{\partial u}{\partial x} = f'$$

# La solution exacte peut être discontinue !

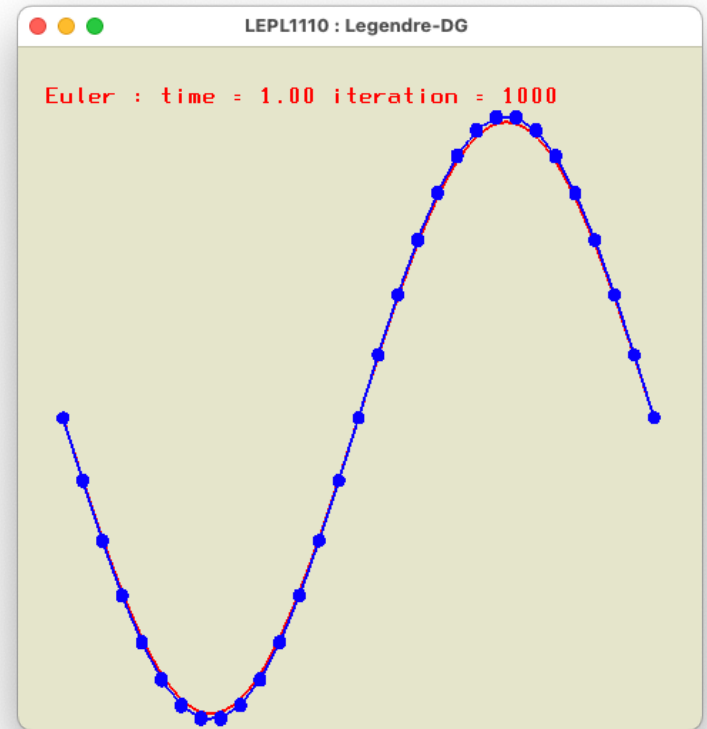


$$u(t, x) \approx u_e^h(t, x) = \sum_{i=0}^p U_i^e(t) \phi_i(x)$$

# Avec peu d'éléments linéaires...

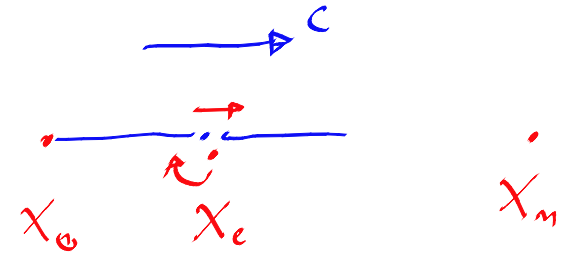


... ou beaucoup !



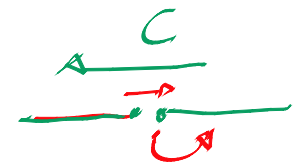
# La méthode DG...

$$\sum_{j=0}^p \langle \phi_i \frac{\partial u_h}{\partial t} \rangle_e + \langle \phi_i \frac{\partial c u_e^h}{\partial x} \rangle_e = 0$$



En effectuant une intégration par partie,

$$\sum_{j=0}^p \underbrace{\langle \phi_i \phi_j \rangle_e}_{A_{ij}^e} \frac{dU_j^e}{dt} = \overbrace{\langle \left( c \frac{d\phi_i}{dx} \right) u_e^h \rangle_e}^{B_i^e(u^h)} - \left[ \phi_i c u^* \right]_{X_{e-1}}^{X_e}$$



En multipliant par l'inverse de la matrice de masse,

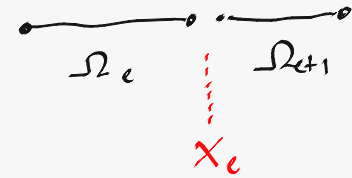
$$\frac{dU_i^e}{dt} = \underbrace{\sum_{j=0}^p A_{ij}^{-1} B_j^e(u^h)}_{F_i^e(u^h)}$$

$$\int_{\Omega_e} \left[ \frac{\partial u_e^h}{\partial t} + \frac{\partial (c u_e^h)}{\partial x} \right] \phi_i = 0 \quad \forall_e$$

$$\int_{\Omega_e} \frac{\partial u_e^h}{\partial t} \phi_i - \int_{\Omega_e} c u_e^h \frac{d\phi_i}{dx} + \underbrace{\left[ (c u_e^h)^* \phi_i \right]_{x_{e-1}}^{x_e}} = 0$$

TERMES  
D'INTERFACE

$(c u_e^h)^*$   
→

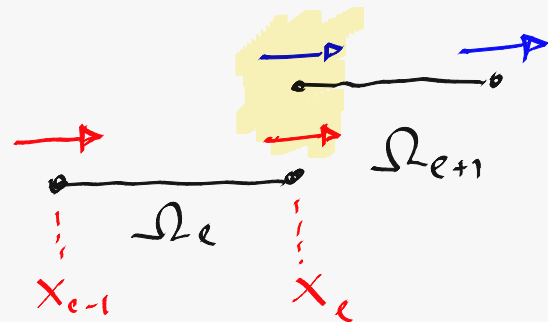


$$\sum_j \int_{\Omega_e} c \frac{d\phi_i}{dx} \phi_j \quad U_j^e$$

$c_{ij}$

$$\sum_j \int_{\Omega_e} \phi_i \phi_j \frac{dU_j^e}{dt}$$


$A_{ij}^e$

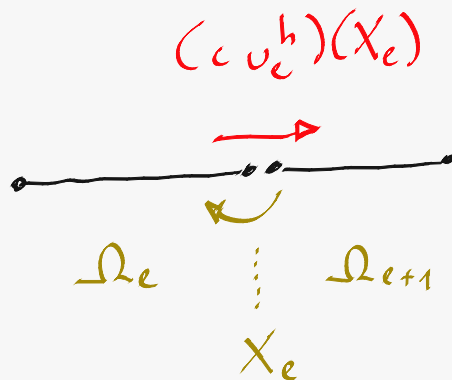


COMMENT  
CALCULER  
 $(c u^h)^*$   
EN  $X_e$  ?

$c > 0$  :-)

$$(c u^h)^* = \left(\frac{1+\zeta}{2}\right) (c u_e^h) + \left(\frac{1-\zeta}{2}\right) (c u_{e+1}^h)$$

  $\zeta = 1$       UPWIND / AMONT  
 $\zeta = 0$       MOYENNE CENTREE  
 $\zeta = -1$       UPWIND A L'ENVERS



# L'idée originale de Lesaint-Raviart...

$$\int_{\Omega_e} \left( \frac{\partial v_e^h}{\partial t} + \frac{\partial}{\partial x} c v_e^h \right) \phi_i + \left[ \underbrace{\left( \frac{1+\zeta}{2} \right)}_{\text{FONCTION PDIDS}} \phi_i(x_{e-1}) (v_e^h(x_{e-1}) - v_{e-1}^h(x_{e-1})) \right] \lambda_c$$

$$+ \left[ \underbrace{\left( \frac{1-\zeta}{2} \right)}_{\text{FONCTION PDIDS}} \phi_i(x_e) (v_{e+1}^h(x_e) - v_e^h(x_e)) \right] \lambda_c = 0$$

$\zeta = 1$  UPWIND  
 $\zeta = 0$  CENTRÉ :-)

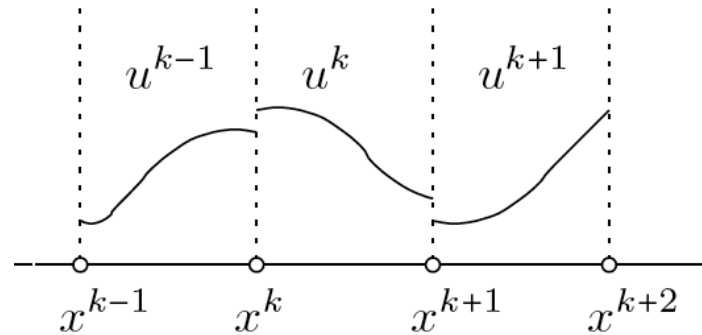
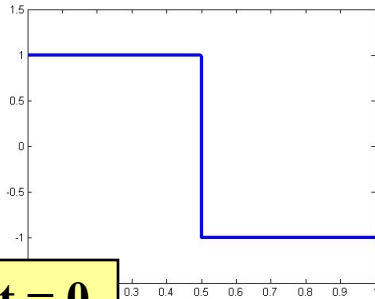
DG = PETROV-GALERKIN  
APPLIQUÉ AUX  
CONDITIONS DE  
RACCORD !

Lesaint, P., & Raviart, P. A. (1974).  
On a finite element method for solving the neutron transport equation.  
Publications mathématiques et informatique de Rennes, (S4), 1-40



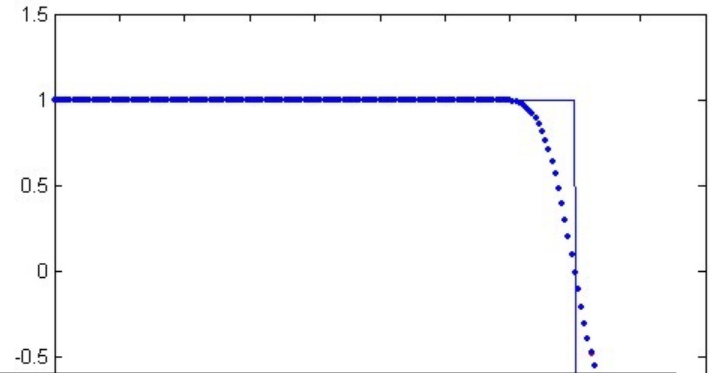
# The Discontinuous so-called Galerkin Method

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0$$



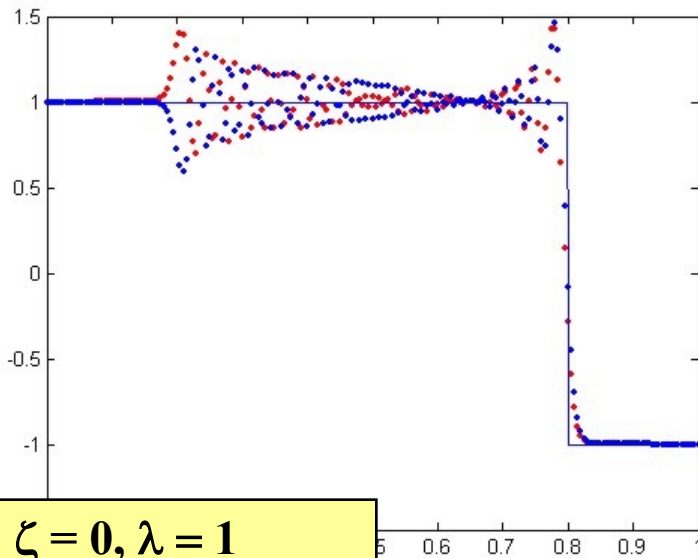
**L'approximation est discontinue !**

# How to impose the continuity constraint ?

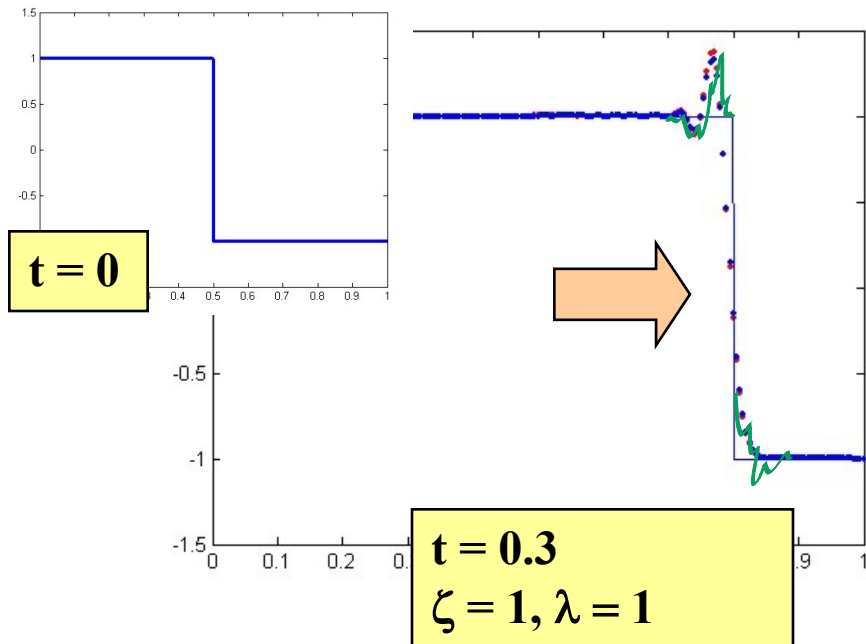


$$\zeta = 1, \lambda = 100$$

Almost the classical upwind difference !  
Stable, but a lot of numerical diffusion...



$$\zeta = 0, \lambda = 1$$

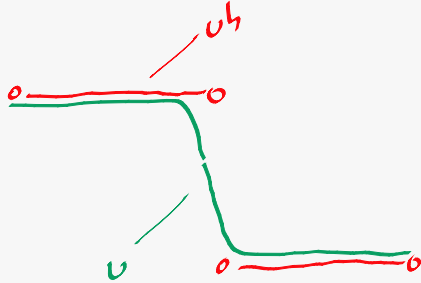


$t = 0$

$t = 0.3$

$$\zeta = 1, \lambda = 1$$

DG



$$c u' = \varepsilon u'' + f$$

DG

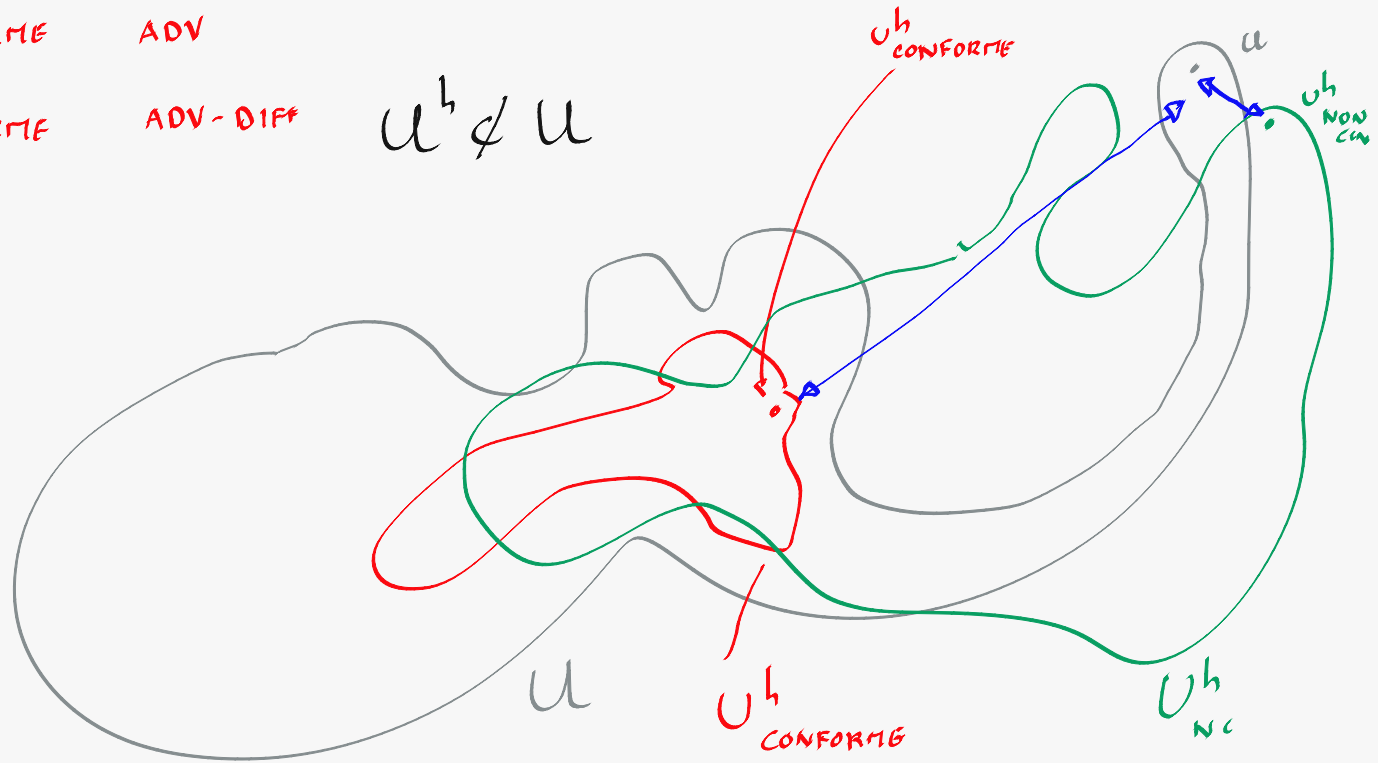
CONFORME

ADV

NON  
CONFORME

ADV-DIFF

$$u^h \neq u$$



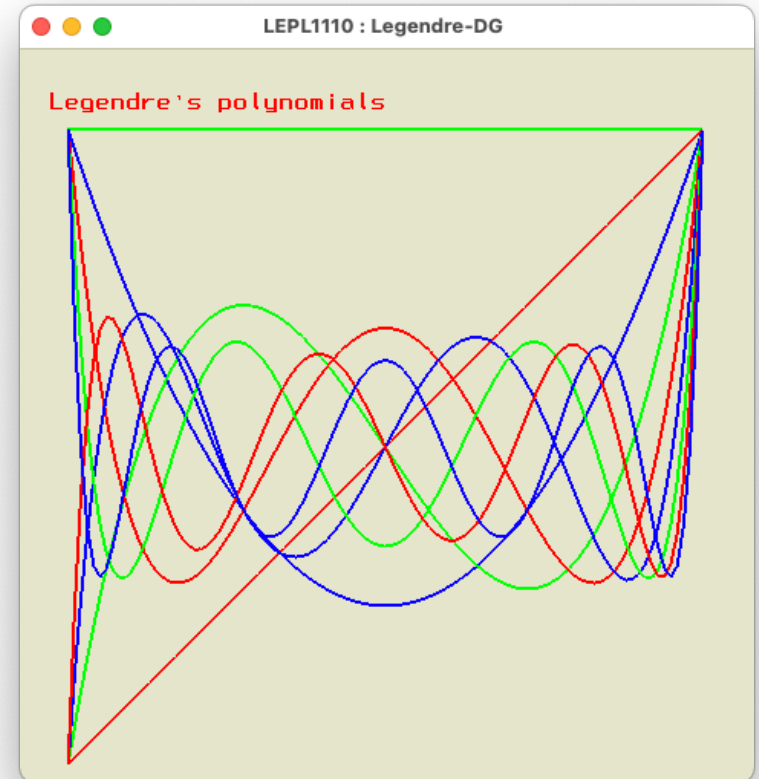
# Des fonctions de base un fifrelin astucieuses...

$$\phi_0(x) = 1$$

$$\phi_1(x) = x$$

...

$$n \phi_n(x) = (2n - 1) x \phi_{n-1}(x) - (n - 1) \phi_{n-2}(x)$$



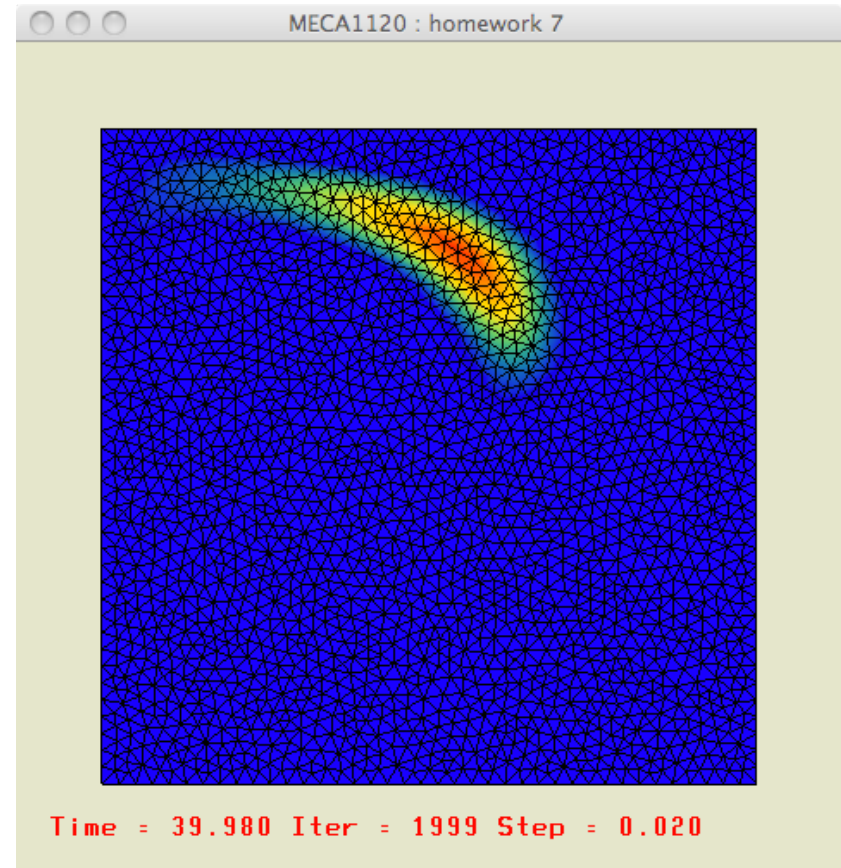
Et les deux matrices locales  
discrètes deviennent  
si simples !

$$\int_{-1}^1 \phi_i \phi_j = \frac{2}{(2p+1)} \delta_{ij}$$

$$\int_{-1}^1 \phi_i' \phi_j = \underbrace{[\phi_i \phi_j]_{-1}^1}_{1 - (-1)^{i+j}} - \underbrace{\int_{-1}^1 \phi_i \phi_j'}_{=0 \text{ PAR ORTHOGONALITÉ}}$$

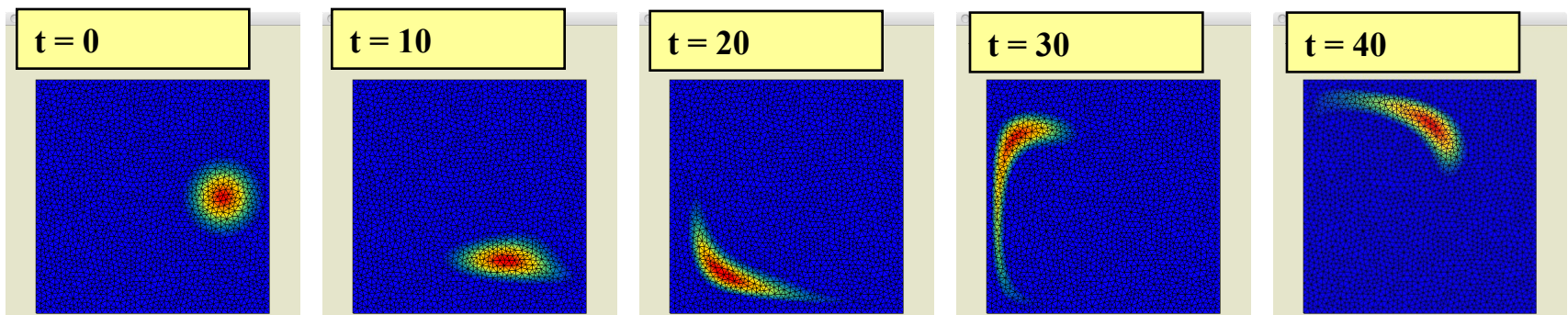
$i > j$

# Discontinuous Galerkin Method



# Implémentation de la méthode DG...

$$\frac{\partial c}{\partial t} + \frac{\partial}{\partial x}(uc) + \frac{\partial}{\partial y}(vc) = 0$$



$$\frac{\partial c}{\partial t} + \frac{\partial}{\partial x}(uc) + \frac{\partial}{\partial y}(vc) = 0$$

$$c(t, x, y) \approx c_e^h(t, x, y) \approx \sum_{i=1}^3 C_i^e(t) \phi_i(x, y)$$

$$\sum_{j=1}^3 \underbrace{\langle \phi_i \phi_j \rangle_e}_{A_{ij}^e = \hat{A}_{ij} J_e} \frac{dC_j^e}{dt} = \overbrace{\langle \left( u \frac{\partial \phi_i}{\partial x} + v \frac{\partial \phi_i}{\partial y} \right) c_e^h \rangle_e}_{B_i^e(c^h)} + \ll \phi_i \beta c^* \gg_e$$

En multipliant par l'inverse de la matrice de masse,

$$\frac{dC_i^e}{dt} = \underbrace{\sum_{j=1}^3 \hat{A}_{ij}^{-1} B_j^e(c^h) / J_e}_{F_i^e(c^h)}$$

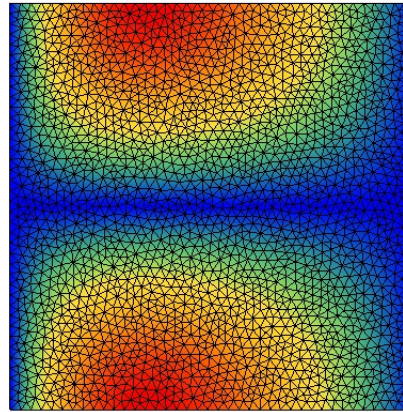
En fonction de la vitesse normale le long d'un segment, on calcule les flux à l'entrée du domaine :

$$\beta = (un_x + vn_y)$$

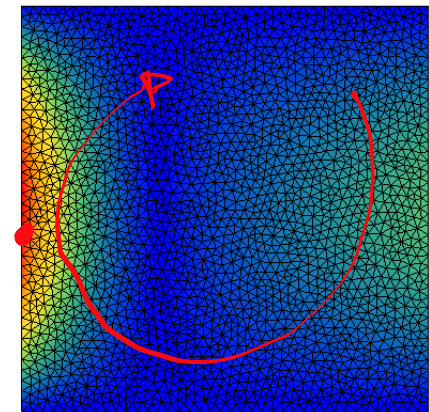
# Equations discrètes



# Un petit gyre océanique bien connu...



$\max(u) = 0.0548$



$\max(v) = 0.1512$

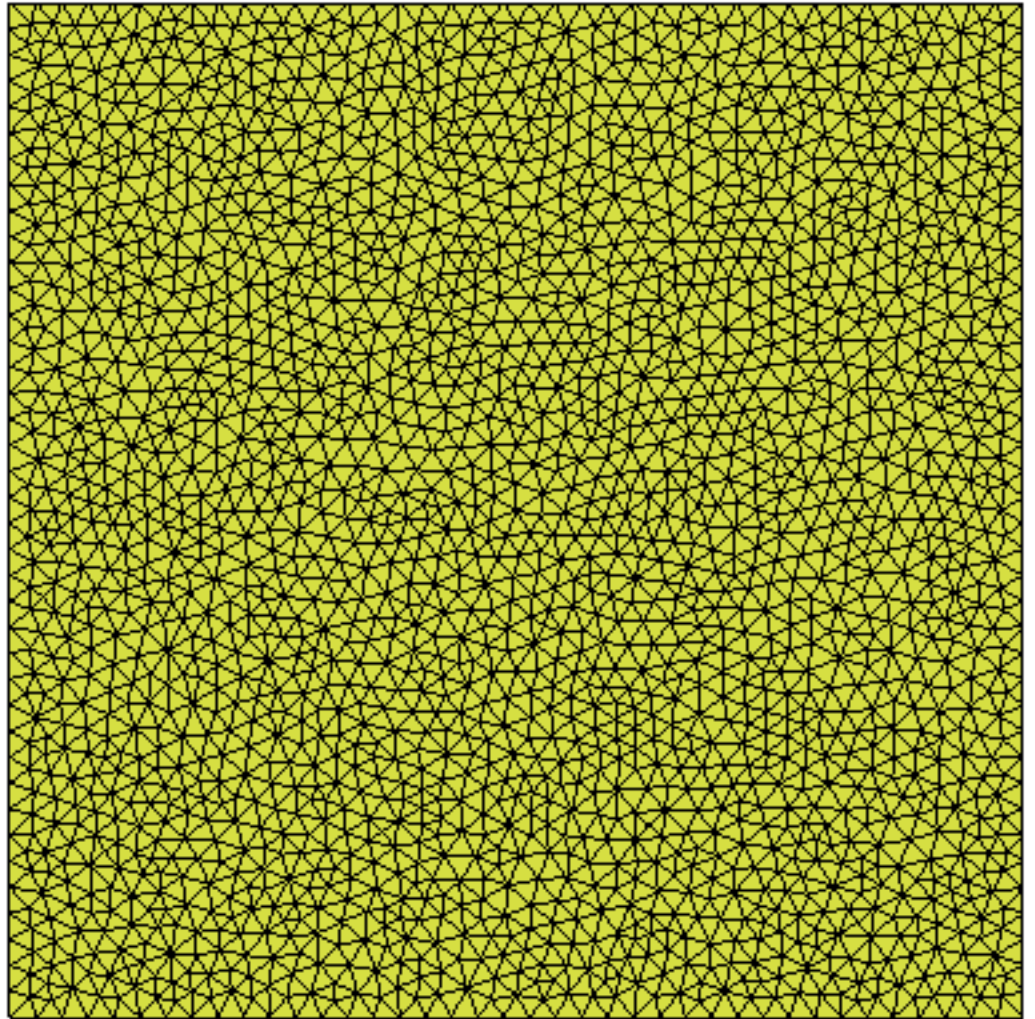
```
const double tau0 = 0.1;
const double L = 1e6;
const double gamm = 1e-6;
const double rho = 1000;
const double delta = 1;
const double g = 9.81;
const double h = 1000;
const double f0 = 1e-4;
const double beta = 0.5e-11;
```

```
double Y = y - 0.5;
double epsilon = gamm / (L * beta);
double Z1 = (-1 + sqrt(1 + (2 * M_PI * delta * epsilon) * (2 * M_PI * delta * epsilon))) / (2 * epsilon);
double Z2 = (-1 - sqrt(1 + (2 * M_PI * delta * epsilon) * (2 * M_PI * delta * epsilon))) / (2 * epsilon);
double D = ((exp(Z2) - 1) * Z1 + (1 - exp(Z1)) * Z2) / (exp(Z1) - exp(Z2));
double f1 = M_PI / D * (1 + ((exp(Z2) - 1) * exp(x * Z1) + (1 - exp(Z1)) * exp(x * Z2)) / (exp(Z1) - exp(Z2)));
double f2 = 1 / D * ((exp(Z2) - 1) * Z1 * exp(x * Z1) + (1 - exp(Z1)) * Z2 * exp(x * Z2)) / (exp(Z1) - exp(Z2));
```

```
u[0] = D * tau0 / (M_PI * gamm * rho * h) * f1 * sin(M_PI * Y);
v[0] = D * tau0 / (M_PI * gamm * rho * delta * h) * f2 * cos(M_PI * Y);
```

**Comme évaluer cela prend beaucoup de temps, on va calculer une approximation linéaire discontinue de la vitesse !**

Un joli  
maillage...



Taille typique des éléments : 0.016 ... 0.032

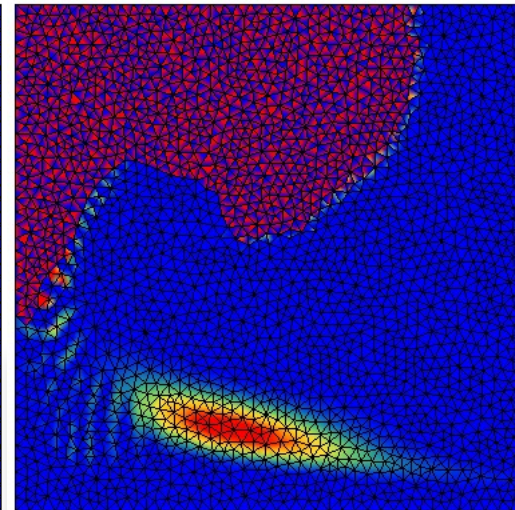
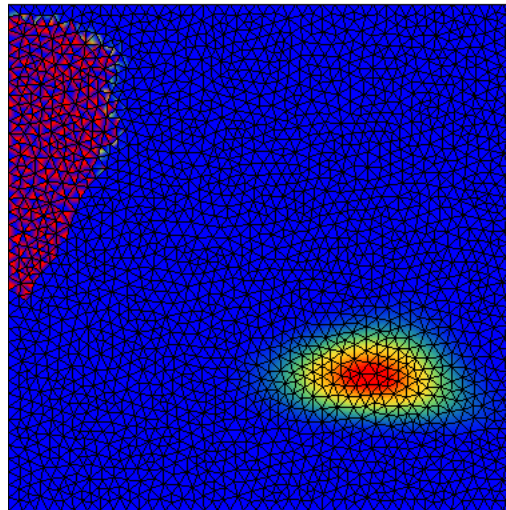
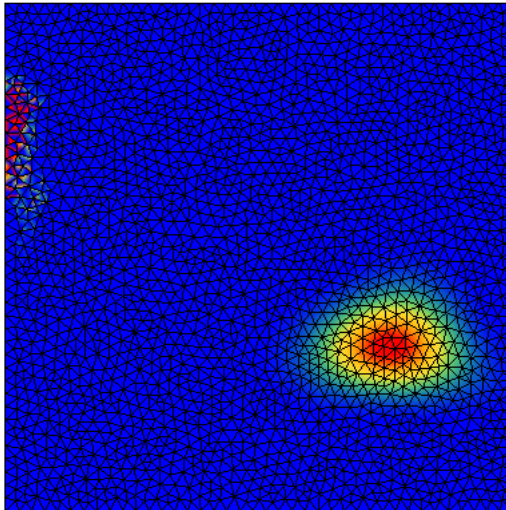
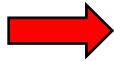
# Un pas de temps critique...

$$\Delta t \leq \frac{h}{\beta}$$



**Pas de temps = 0.8 :  
Boum !**

Endroit  
le plus  
critique !

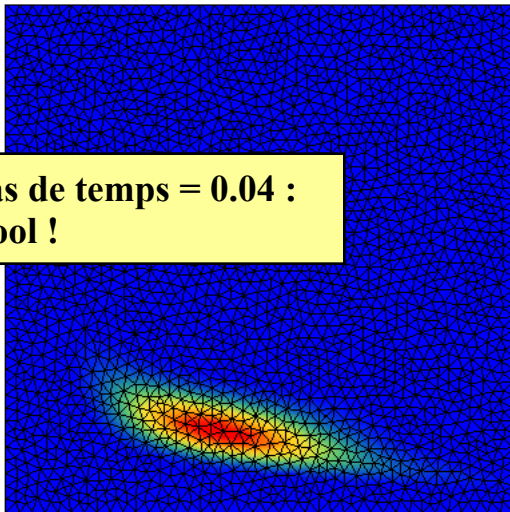


**Plus petit élément : 0.016  
Vitesse maximale : 0.152  
Le plus petit pas de temps : 0.10 environ :-)**

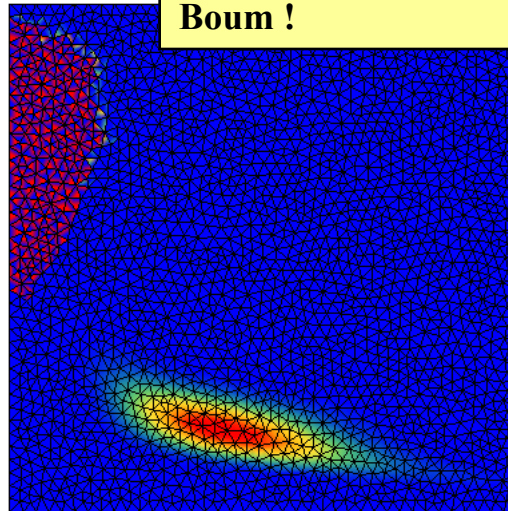
# Diminuous le pas de temps...

$$\Delta t \leq \frac{h}{\beta}$$

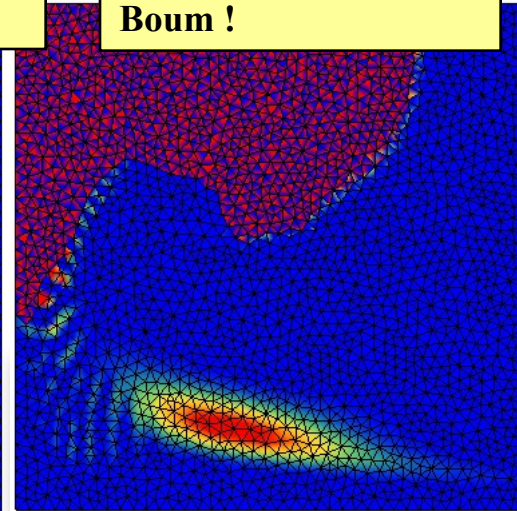
**Pas de temps = 0.04 :  
Cool !**



**Pas de temps = 0.06 :  
Boum !**



**Pas de temps = 0.08 :  
Boum !**



**Plus petit élément : 0.016  
Vitesse maximale : 0.152  
Le plus petit pas de temps : 0.10 environ :-)**

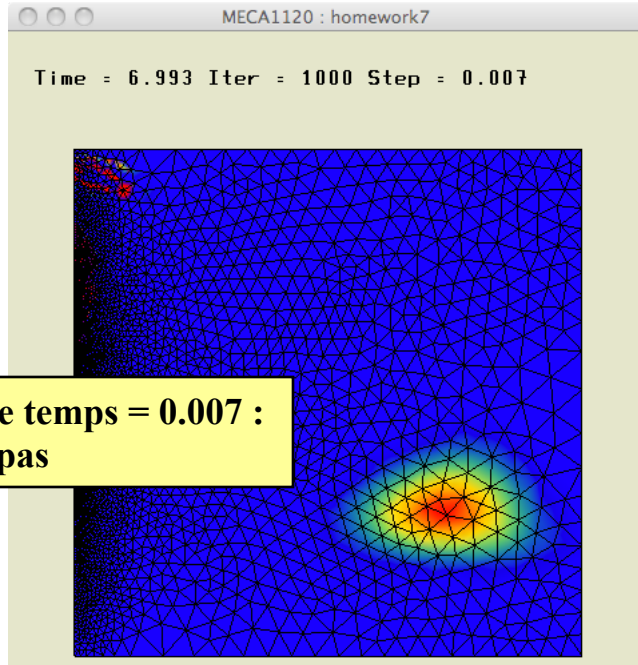
# The time stepping issue

**C'est le plus petit élément qui est contraignant !  
Très très ennuyeux !**

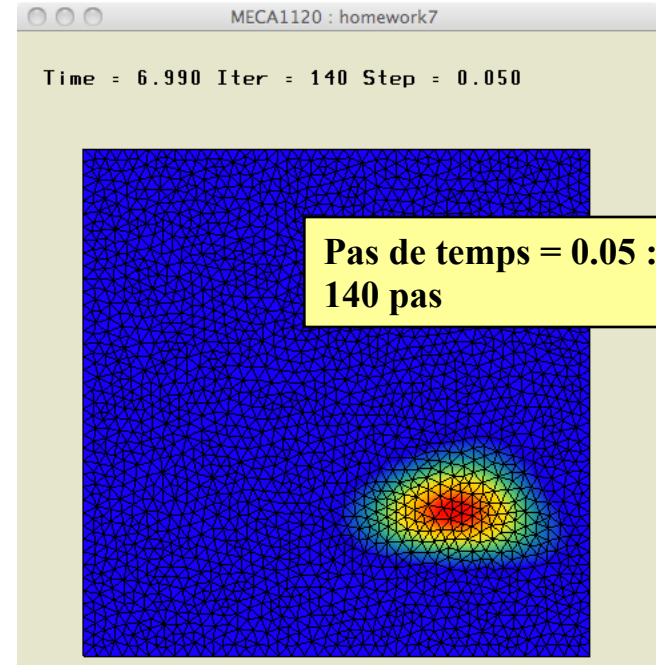
**Endroit  
le plus  
critique !**



**Pas de temps = 0.007 :  
1000 pas**



**Pas de temps = 0.05 :  
140 pas**

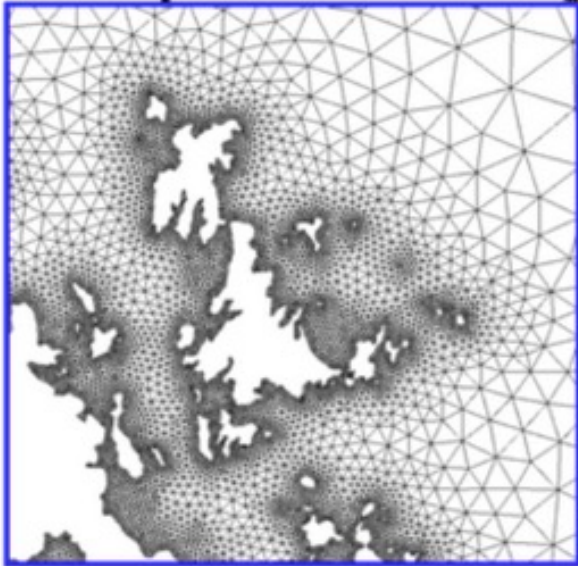
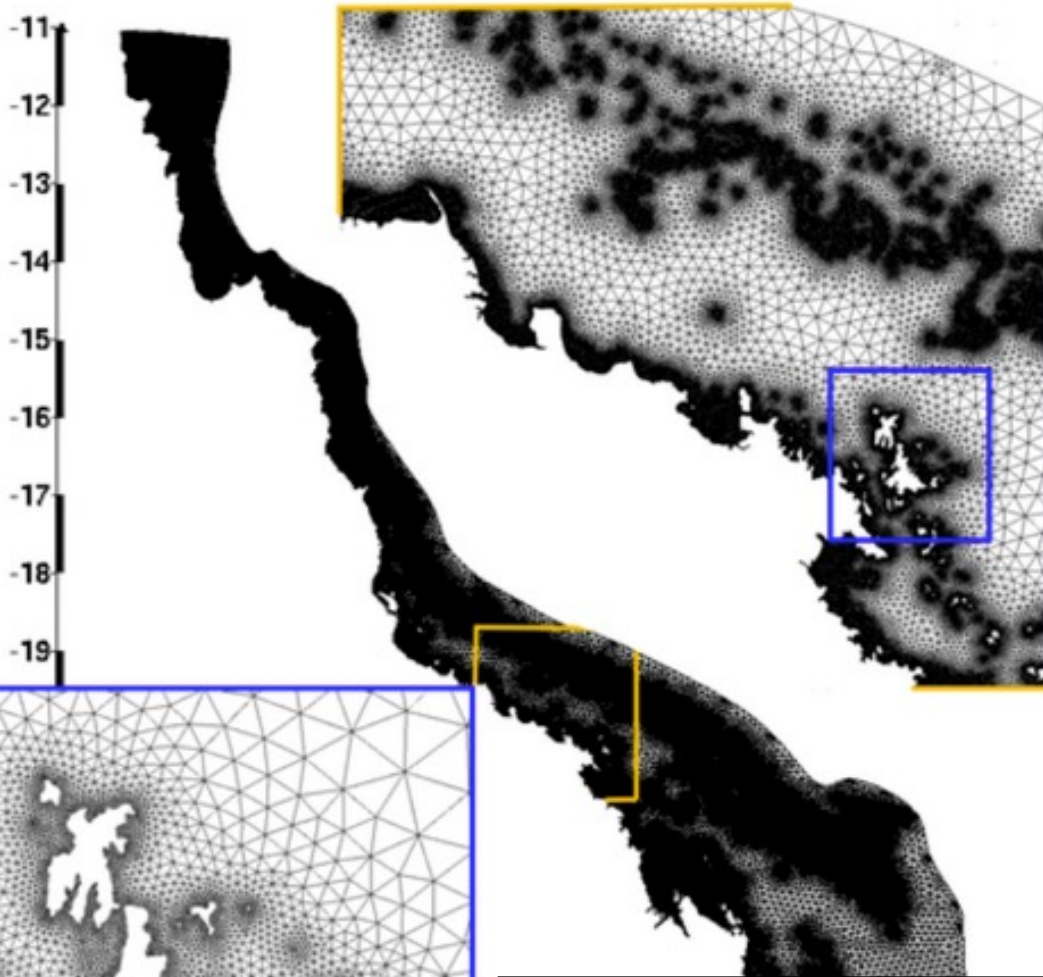


# The time stepping issue

- *890,000 triangles*
- *Smallest element : 7 m*
- *Largest element : 3,300 m*
- *99.9 % > 60m*

**The time step is constrained by the smallest element .**

- **Use innovative time stepping procedures**
- **Implicit-explicit (IMEX) schemes**
- **Multirate schemes (Bruno Seny)**

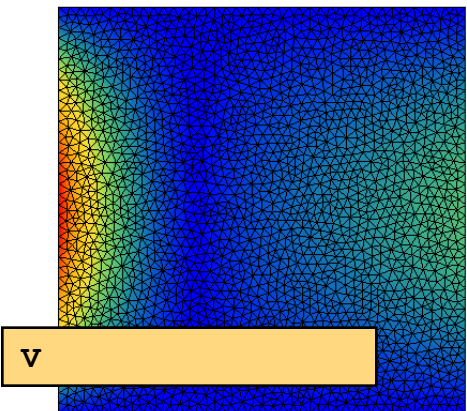
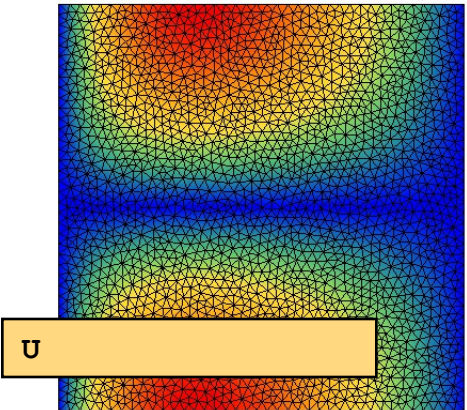
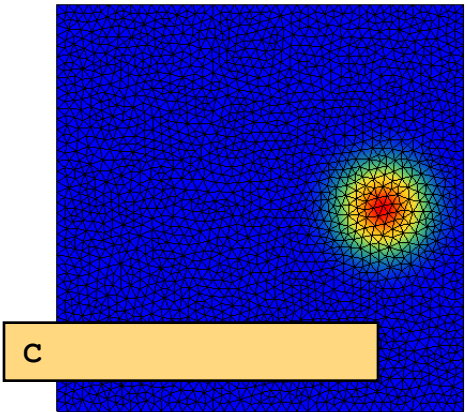


# Notre problème...

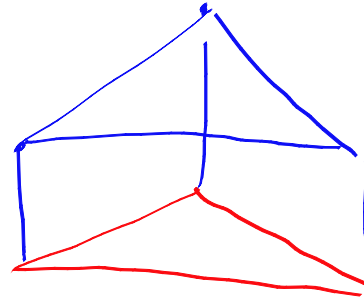
```
typedef struct {  
    femMesh *mesh;  
    femEdges *edges;  
    ...  
    int size;  
    double *C;  
    double *U;  
    double *V;  
    ...  
} femAdvectionProblem;
```

Ces trois champs sont initialisés avec une approximation au sens des moindres carrés L2

```
int myProblem->size = myProblem->mesh->nElem * nLocal + 1;
```



Et calculons aussi  
l'inverse de  
la matrice  
de masse

 $\hat{A}_{ij}$ 

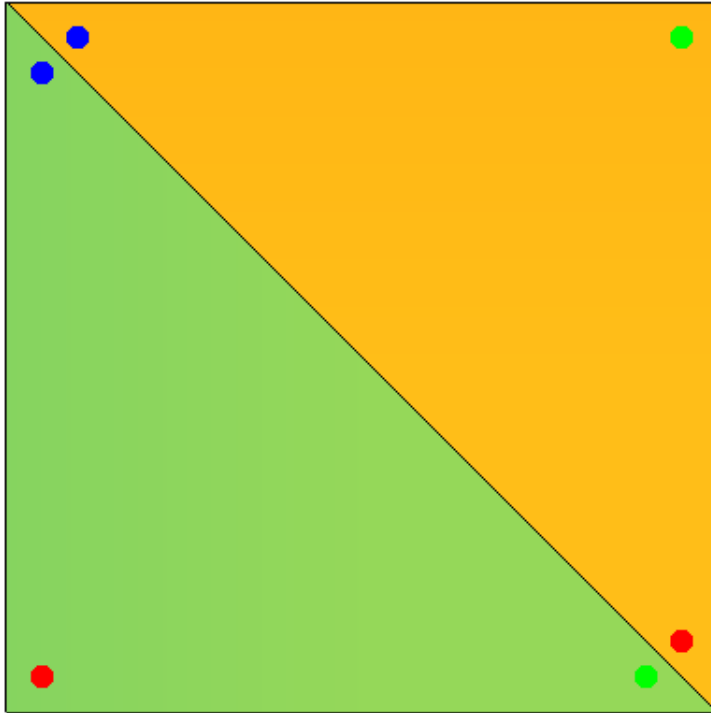
+8.3333333333e-02	+4.1666666667e-02	+4.1666666667e-02
+4.1666666667e-02	+8.3333333333e-02	+4.1666666667e-02
+4.1666666667e-02	+4.1666666667e-02	+8.3333333333e-02

 $\hat{A}_{ij}^{-1}$ 

+1.8000000000e+01	-6.0000000000e+00	-6.0000000000e+00
-6.0000000000e+00	+1.8000000000e+01	-6.0000000000e+00
-6.0000000000e+00	-6.0000000000e+00	+1.8000000000e+01



# Des triangles...



Nombre d'éléments :  $n = 2$

Number of nodes 4			
0 :	0.0000000e+00	0.0000000e+00	
1 :	1.0000000e+00	0.0000000e+00	
2 :	0.0000000e+00	1.0000000e+00	
3 :	1.0000000e+00	1.0000000e+00	
Number of triangles 2			
0 :	0	1	2
1 :	1	3	2

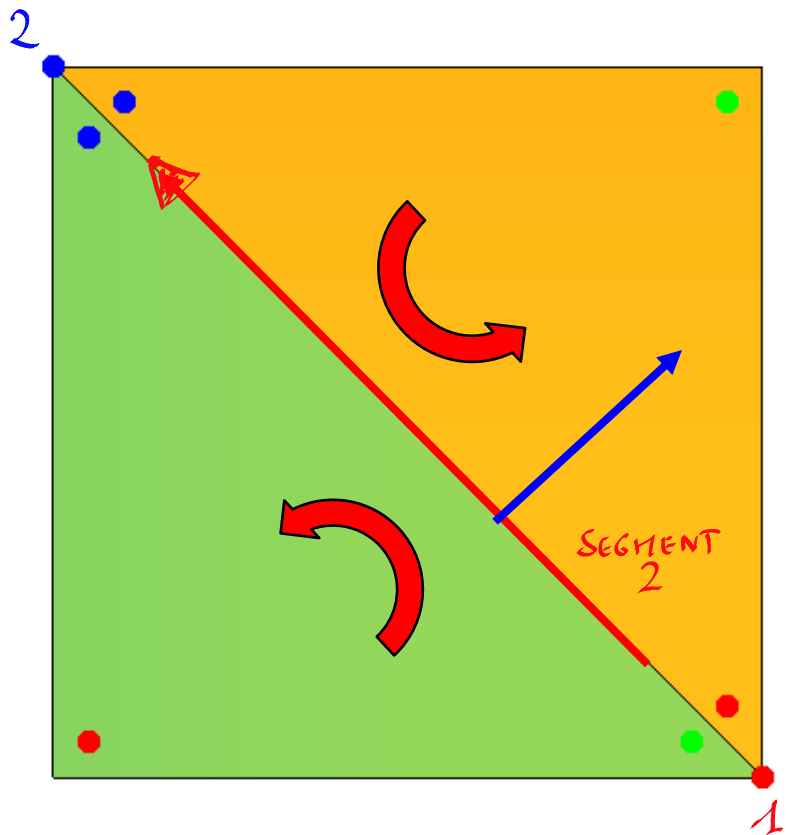
```
double *C = theProblem->soluce;  
double a = 0.4;  
double b = 0.7;  
  
C[0] = a;  
C[1] = a;  
C[2] = a;  
C[3] = b;  
C[4] = b;  
C[5] = b;  
C[6] = 0;
```

**Approximation linéaire discontinue**

**Nombre de degrés de liberté intérieur :  $3n$  !**  
**On ajoute un degré de liberté pour l'extérieur**

**On a un vecteur de taille  $3n+1$**

# ...et des segments



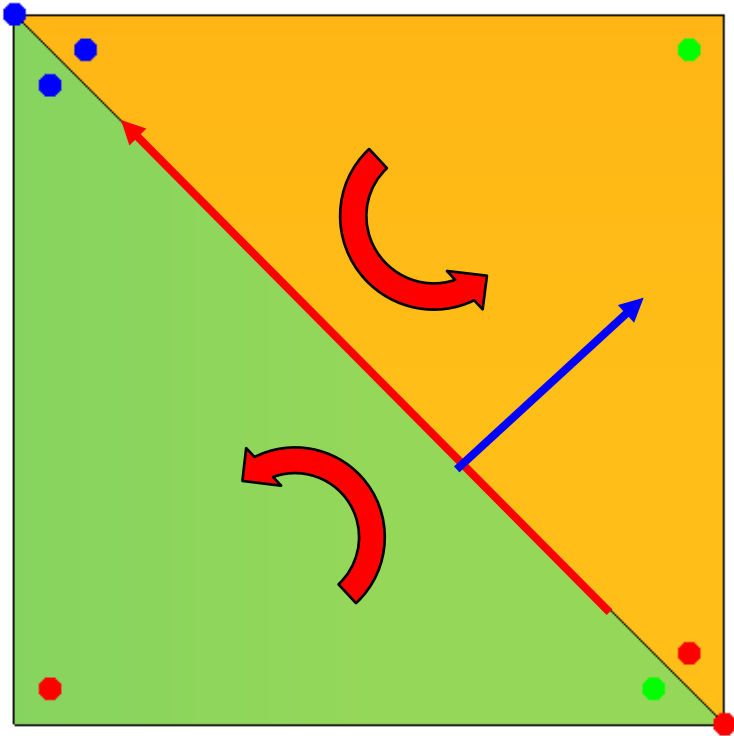
Number of nodes 4		
0 :	0.0000000e+00	0.0000000e+00
1 :	1.0000000e+00	0.0000000e+00
2 :	0.0000000e+00	1.0000000e+00
3 :	1.0000000e+00	1.0000000e+00
Number of triangles 2		
0 :	0	1 2
1 :	1	3 2

0 :	3	2 :	1	-1
1 :	1	3 :	1	-1
2 :	1	2 :	0	1
3 :	2	0 :	0	-1
4 :	0	1 :	0	-1

Sommets                      Eléments

**L'orientation est en accord avec le premier élément et opposée au second !**

# Obtenir les indices des degrés de liberté



Number of nodes 4			
0 :	0.0000000e+00	0.0000000e+00	
1 :	1.0000000e+00	0.0000000e+00	
2 :	0.0000000e+00	1.0000000e+00	
3 :	1.0000000e+00	1.0000000e+00	
Number of triangles 2			
0 :	0	1	2
1 :	1	3	2

0 :	3	2 :	1	-1
1 :	1	3 :	1	-1
2 :	1	2 :	0	1
3 :	2	0 :	0	-1
4 :	0	1 :	0	-1

```
femAdvTriangleMap(myProblem, 1, map) ;
```



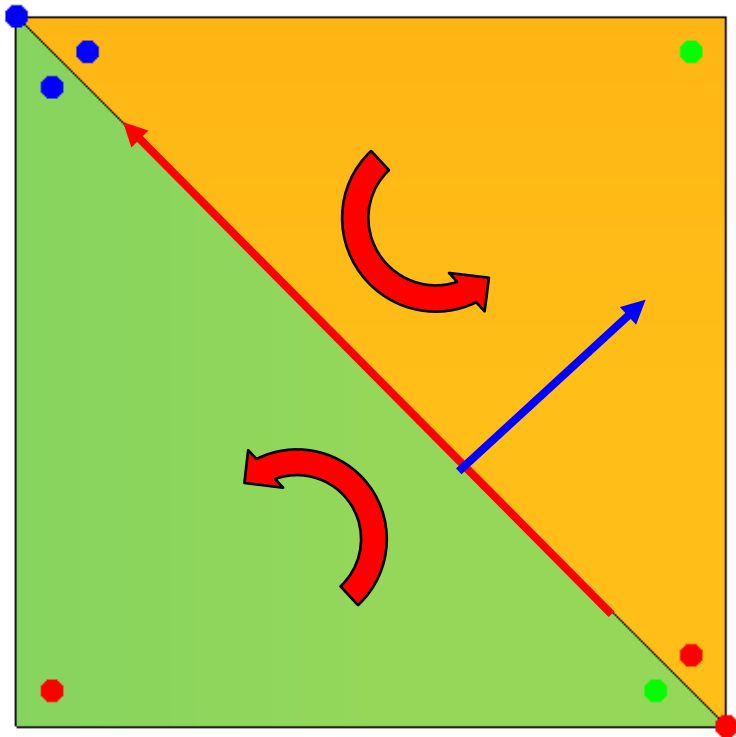
```
map[3] = {3, 4, 5};
```

```
femAdvTriangleMap(myProblem, 0, map) ;
```



```
map[3] = {0, 1, 2};
```

# Obtenir les indices des degrés de liberté



Number of nodes 4			
0 :	0.0000000e+00	0.0000000e+00	
1 :	1.0000000e+00	0.0000000e+00	
2 :	0.0000000e+00	1.0000000e+00	
3 :	1.0000000e+00	1.0000000e+00	
Number of triangles 2			
0 :	0	1	2
1 :	1	3	2

0 :	3	2 :	1	-1
1 :	1	3 :	1	-1
2 :	1	2 :	0	1
3 :	2	0 :	0	-1
4 :	0	1 :	0	-1

```
femAdvEdgeMap (myProblem, 2, map) ;
```



```
map[2][2] = {{1,2};{3,5}};
```

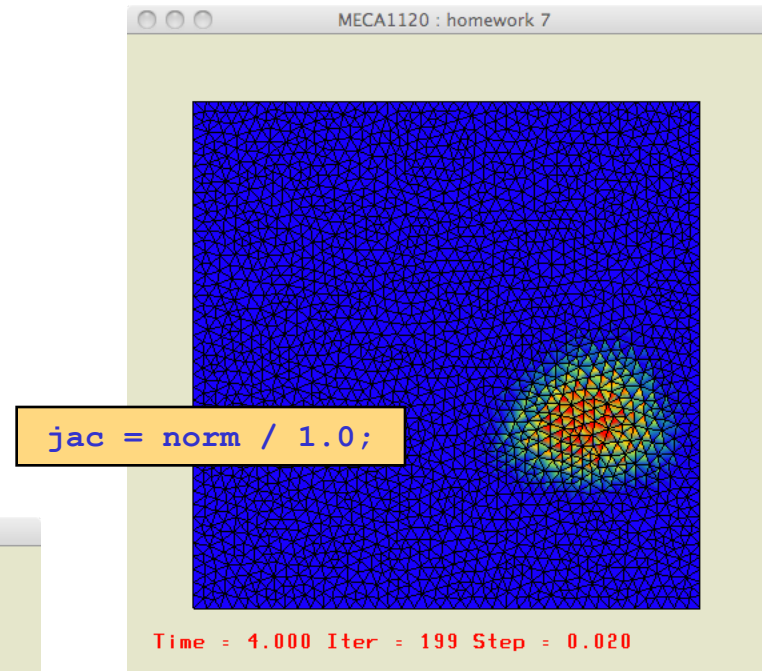
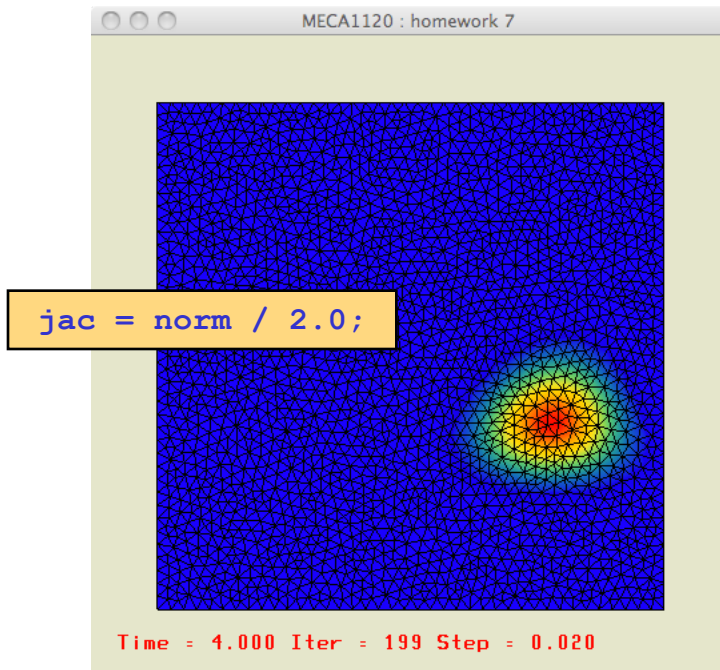
```
femAdvEdgeMap (myProblem, 4, map) ;
```



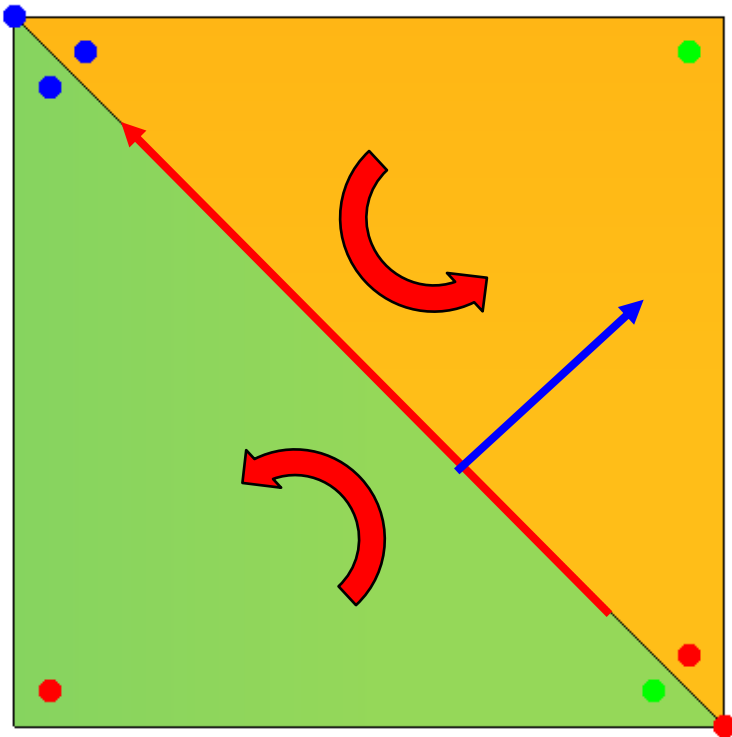
```
map[2][2] = {{0,1};{6,6}};
```

1  
ELEMENT  
C<sub>0</sub>  
C<sub>1</sub>  
C<sub>2</sub>  
C<sub>3</sub>  
C<sub>4</sub>  
C<sub>5</sub>  
C<sub>6</sub>  
ELEM  
2

Help...  
Professor  
It blows up !



# Commencer par le petit maillage !



```
Number of nodes 4
0 : 0.0000000e+00 0.0000000e+00
1 : 1.0000000e+00 0.0000000e+00
2 : 0.0000000e+00 1.0000000e+00
3 : 1.0000000e+00 1.0000000e+00
```

```
Number of triangles 2
0 : 0 1 2
1 : 1 3 2
```

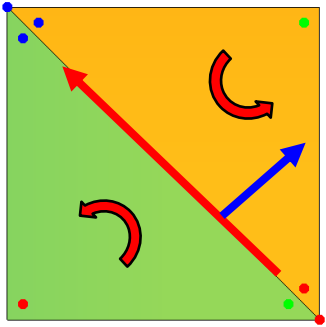
```
0 : 3 2 : 1 -1
1 : 1 3 : 1 -1
2 : 1 2 : 0 1
3 : 2 0 : 0 -1
4 : 0 1 : 0 -1
```

```
map = dg.edgeMap(myProblem,2)
```

```
map = [[1,2],[3,5]]
```

```
map = dg.edgeMap(myProblem,4)
```

```
map = [[0,1],[[]]]
```

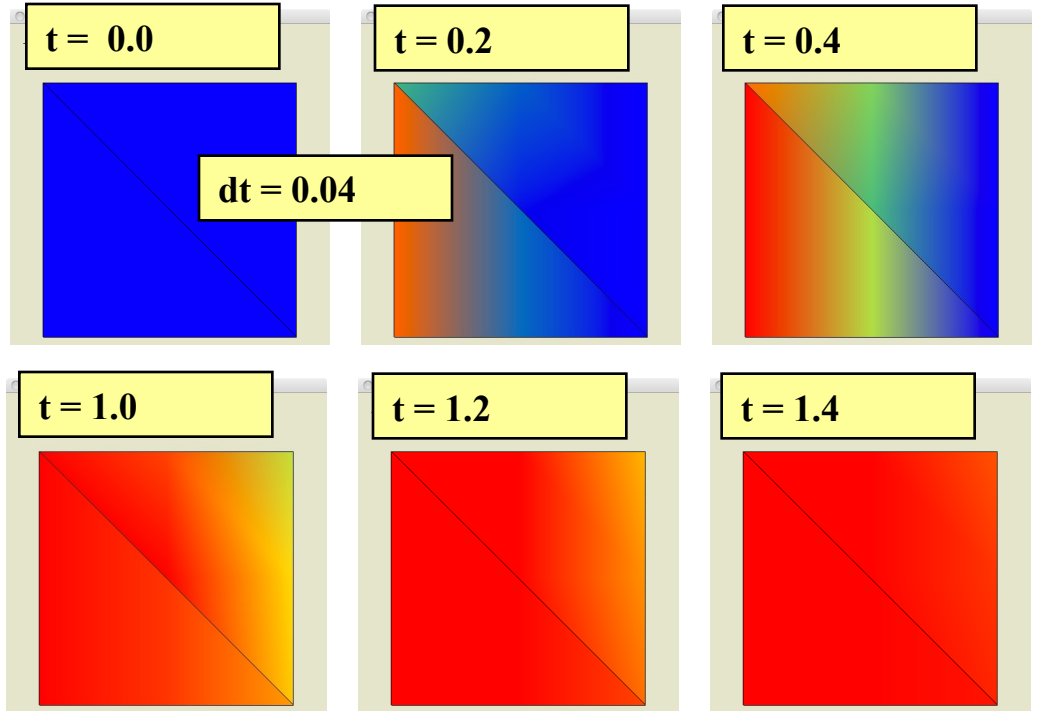


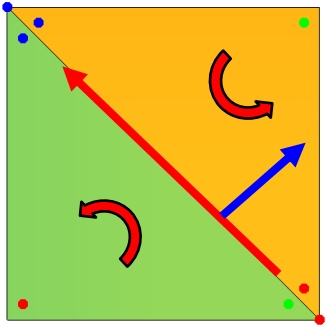
$\rightarrow U = 1$

$U = 1$   
 $V = 0$   
 $C = 0$

Un problème  
 tout simple  
 tout d'abord

$\rightarrow$





```

advectionAddIntegralsTriangles (theProblem) ;
advectionAddIntegralsEdges (theProblem) ;
advectionMultiplyInverseMatrix (theProblem) ;

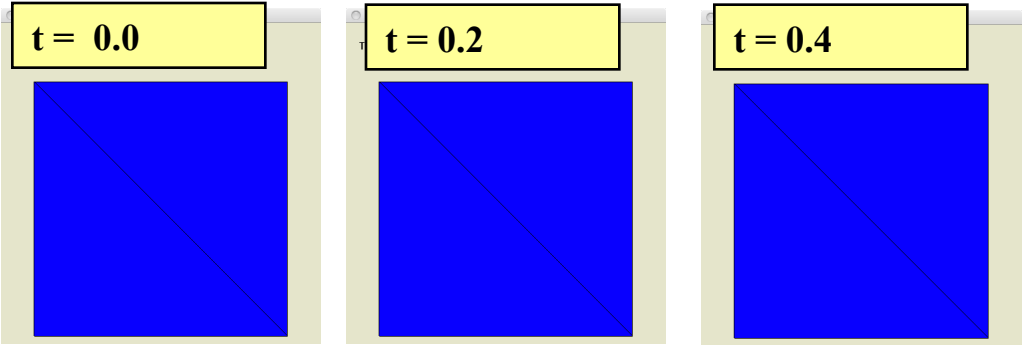
```

**U = 1**  
**V = 0**  
**C = 0**

```

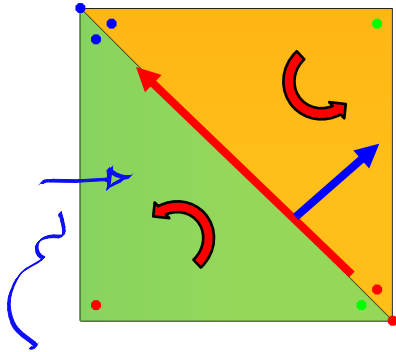
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
-- after triangles
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
-- after edges

```



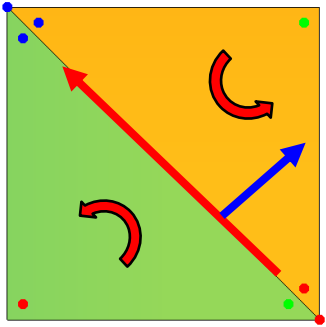
**Sanity  
 check #1**





IMPOSER  
UN FLUX  
ENTRANT

Comment imposer  
un flux de matière  
sur le côté gauche ?



```

advectionAddIntegralsTriangles (theProblem) ;
advectionAddIntegralsEdges (theProblem) ;
advectionMultiplyInverseMatrix (theProblem) ;

```

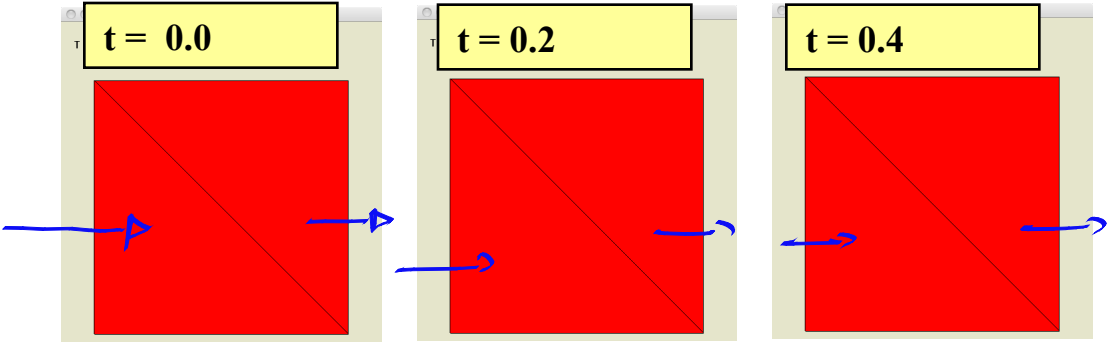
Flux à gauche = 1

U = 1  
V = 0  
C = 1

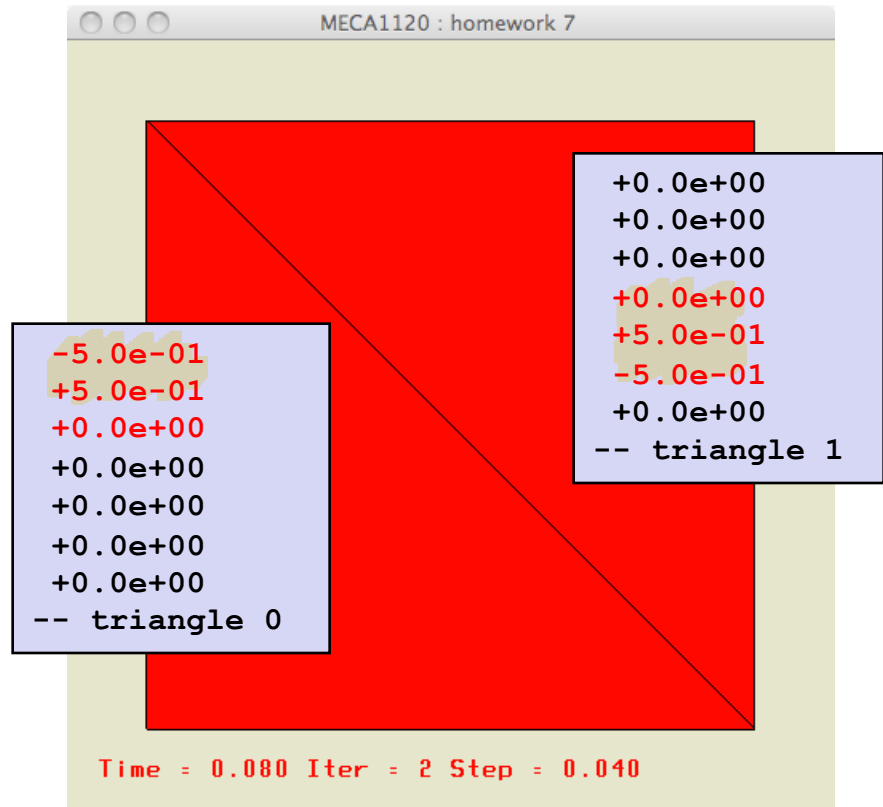
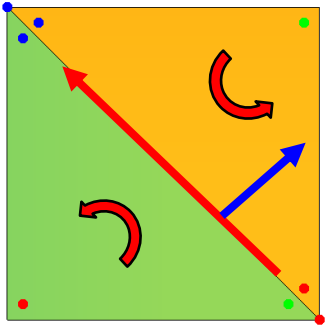
```

+5.0e-01
-5.0e-01
-2.8e-17
+2.8e-17
-5.0e-01
+5.0e-01
+4.4e-16
-- after edges
-1.1e-15
+1.0e-15
-2.8e-17
+2.8e-17
+9.2e-16
-1.1e-15
+4.4e-16
-- after triangles

```



Sanity  
check #2



Number of nodes 4

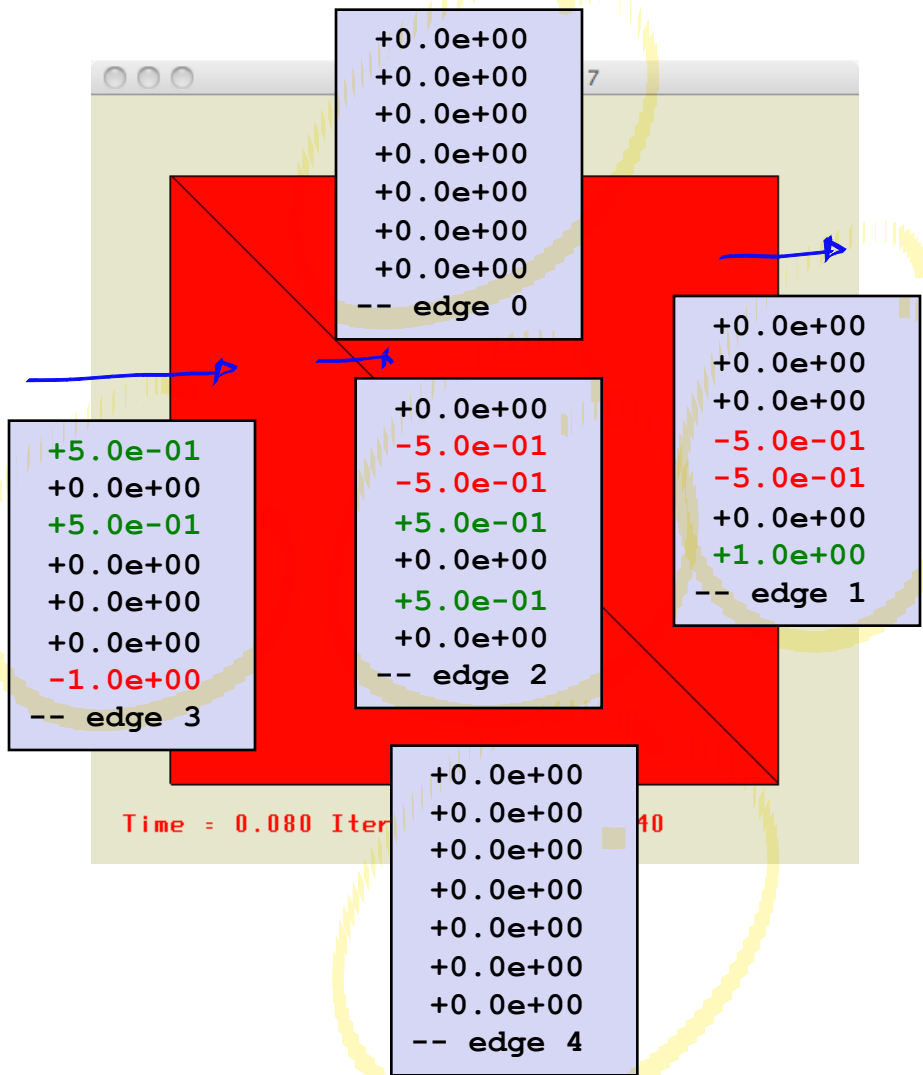
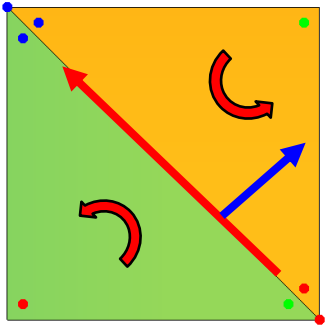
0 :	0.0000000e+00	0.0000000e+00
1 :	1.0000000e+00	0.0000000e+00
2 :	0.0000000e+00	1.0000000e+00
3 :	1.0000000e+00	1.0000000e+00

Number of triangles 2

0 :	0	1	2
1 :	1	3	2

0 :	3	2 :	1	-1
1 :	1	3 :	1	-1
2 :	1	2 :	0	1
3 :	2	0 :	0	-1
4 :	0	1 :	0	-1

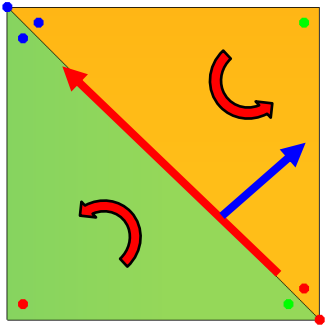
Sanity  
check #2



Number of nodes 4			
0 :	0.0000000e+00	0.0000000e+00	
1 :	1.0000000e+00	0.0000000e+00	
2 :	0.0000000e+00	1.0000000e+00	
3 :	1.0000000e+00	1.0000000e+00	
Number of triangles 2			
0 :	0	1	2
1 :	1	3	2

0 :	3	2 :	1	-1
1 :	1	3 :	1	-1
2 :	1	2 :	0	1
3 :	2	0 :	0	-1
4 :	0	1 :	0	-1

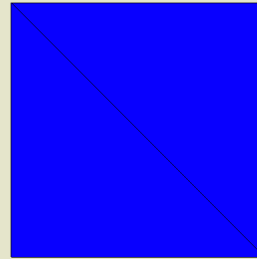
Sanity  
check #2



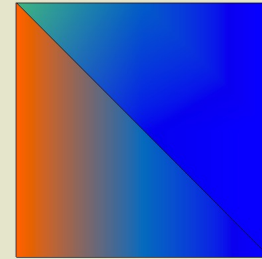
$$\begin{aligned}
 U &= 1 \\
 V &= 0 \\
 C &= 0
 \end{aligned}$$

And now  
and only now

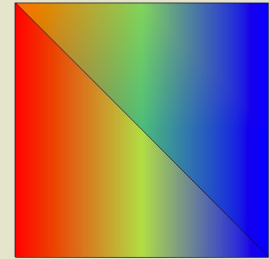
t = 0.0



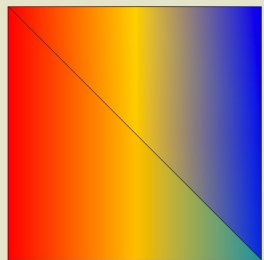
t = 0.2



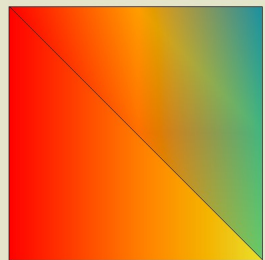
t = 0.4



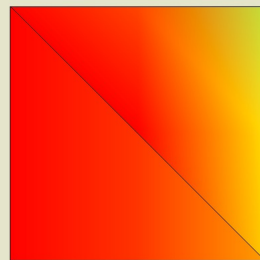
t = 0.6



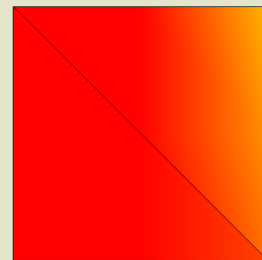
t = 0.8



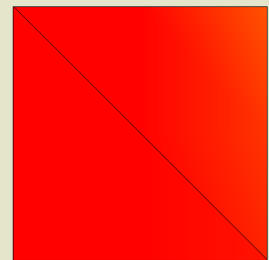
t = 1.0



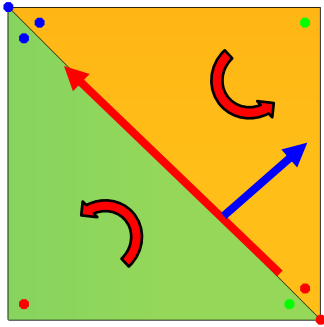
t = 1.2



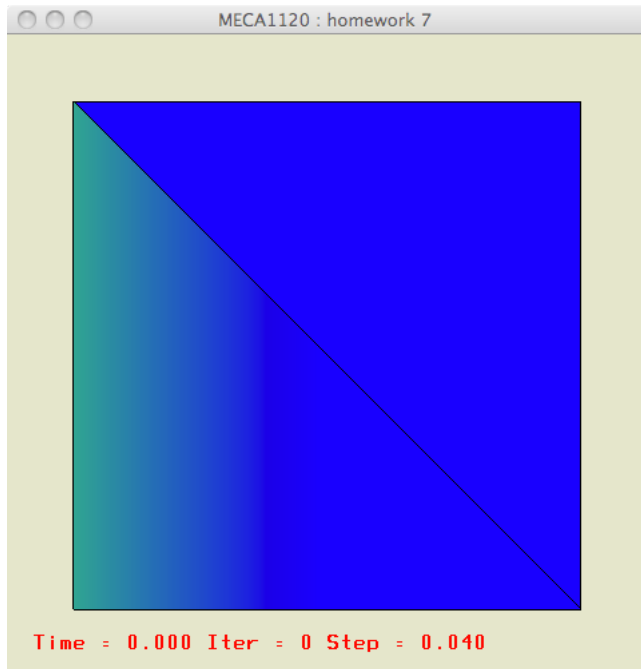
t = 1.4



# Itération one :-)



```
advectionAddIntegralsTriangles (theProblem) ;
advectionAddIntegralsEdges (theProblem) ;
advectionMultiplyInverseMatrix (theProblem) ;
```



$$\left\langle \left( u \frac{\partial \phi_i}{\partial x} + v \frac{\partial \phi_i}{\partial y} \right) c_e^h \right\rangle_e$$

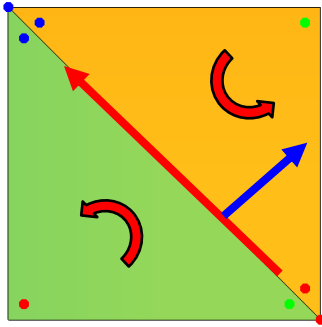
$$\ll \phi_i \beta c^* \gg_e$$

```
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
-- after triangles
+5.0e-01
+0.0e+00
+5.0e-01
+0.0e+00
+0.0e+00
+0.0e+00
-1.0e+00
-- after edges
```

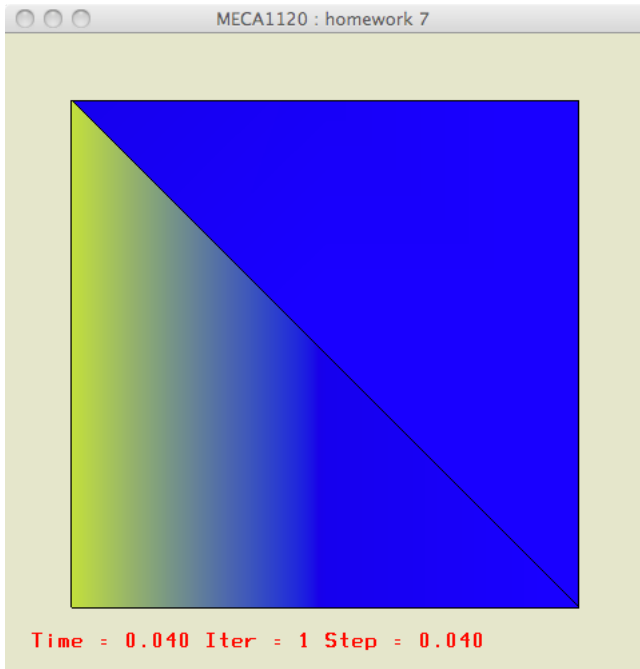
ELEMENT 1

ELEMENT 2

# Itération two :-)



```
advectionAddIntegralsTriangles (theProblem) ;
advectionAddIntegralsEdges (theProblem) ;
advectionMultiplyInverseMatrix (theProblem) ;
```

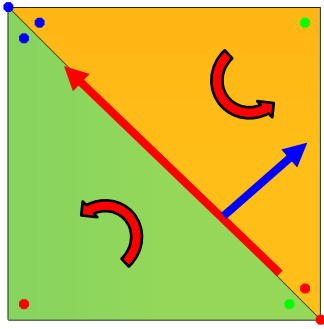


$$\left\langle \left( u \frac{\partial \phi_i}{\partial x} + v \frac{\partial \phi_i}{\partial y} \right) c_e^h \right\rangle_e$$

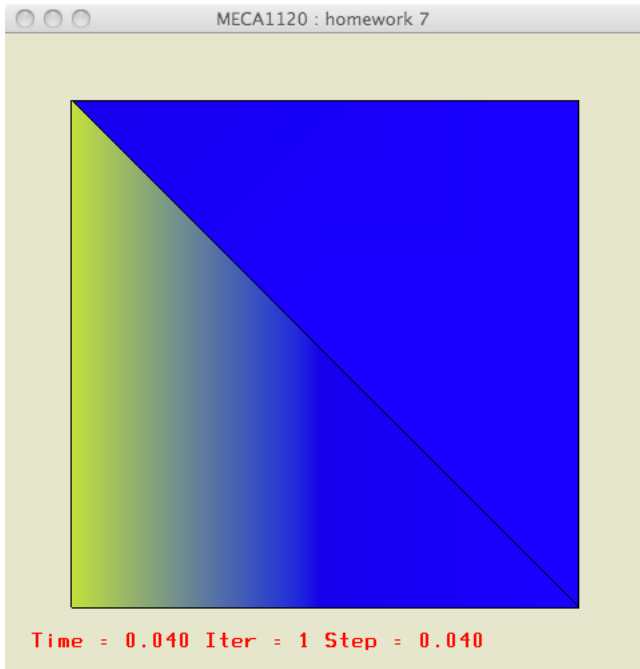
$$\ll \phi_i \beta c^* \gg_e$$

```
-4.0e-02
+4.0e-02
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
+0.0e+00
-- after triangles
+4.6e-01
+8.0e-02
+4.6e-01
-4.0e-02
+0.0e+00
+4.0e-02
-1.0e+00
-- after edges
```

# Itération two :-)



```
advectionAddIntegralsTriangles (theProblem) ;
advectionAddIntegralsEdges (theProblem) ;
advectionMultiplyInverseMatrix (theProblem) ;
```



$$\ll \phi_i \beta c^* \gg_e$$

$$\left\langle \left( u \frac{\partial \phi_i}{\partial x} + v \frac{\partial \phi_i}{\partial y} \right) c_e^h \right\rangle_e$$

```
+5.0e-01
+4.0e-02
+5.0e-01-0.4e-02
-4.0e-02
+0.0e+00
+4.0e-02
-1.0e+00
-- after edges
+4.6e-01
+8.0e-02
+4.6e-01
-4.0e-02
+0.0e+00
+4.0e-02
-1.0e+00
-- after triangles
```



# C'est quoi la solution ?

