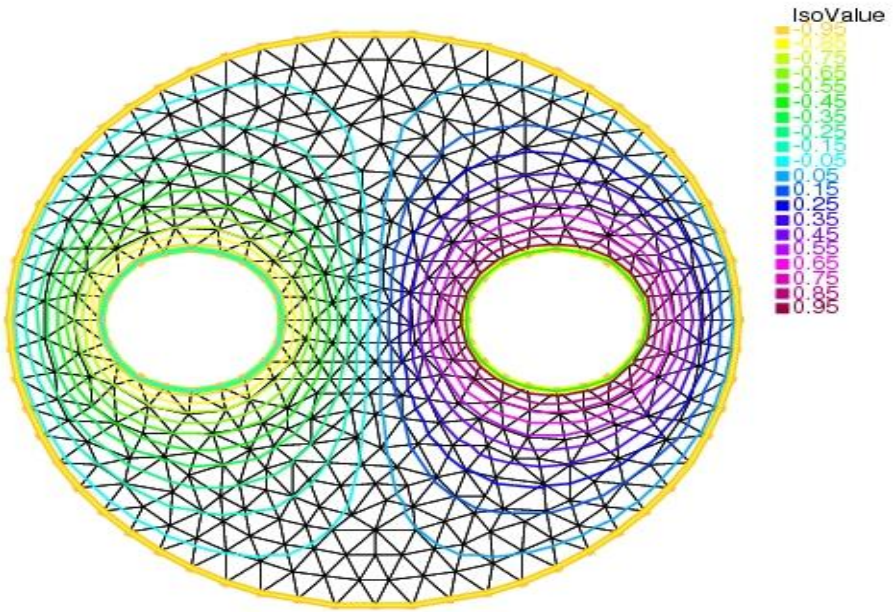
FreeFem++



Introduction sur Freefem++

- Freefem++ est un freeware basé sur C++ porté sous Windows, Unix et Mac OS et dédié à la résolution des équations aux dérivées partielles (elliptique, parabolique, hyperbolique) en 2d et 3d par des méthodes de type éléments finis P0, P1, P2.

Introduction sur Freefem++

- Il s'exécute en ligne de commande.
- Il permet :

  - De créer des maillages(structurés ou non structurés).

  - De résoudre les EDPs (Linéaires ou non linéaires).

  - De visualiser les résultats (maillages et Solutions).

Introduction sur Freefem++

- En général, les problèmes impliquant des équations aux dérivées partielles en deux ou trois dimensions nécessitent l'interpolation des champs inconnus sur des maillages qu'il faut pouvoir manipuler au sein d'un unique programme. C'est ce que propose FreeFem++ en incluant des algorithmes rapides d'interpolation et un langage de création et de manipulation de maillages

Etapes de résolution sur Freefem++

Après avoir écrit le problème sous une forme variationnelle ;

- Définition du domaine de calcul .
- Définition de l'espace d'éléments finis associé au maillage.
- Définition du problème à résoudre, et la méthode de résolution du système.
- Résolution du problème.
- Représentation du résultat dans le domaine sous forme graphique.

.

Les types de données

Variables globales ou variables réservées

- En Freefem++, il existe un certain nombre de variables globales dont voici les plus courantes :
- – **x**, **y** et **z** : les coordonnées du point courant. Pour l'instant, **z** n'est pas encore utilisable, mais il est réservé pour un usage futur.
- – **label** : le numéro de référence de la frontière dans laquelle se situe le point courant.

- – **N** : le vecteur normal sortant unitaire au point courant s'il se situe sur une frontière.

- – **cin**, **cout** et **endl** : les commandes d'affichage /récupération de données issues de C++, utilisées avec << et >>.
- – **pi** : le nombre π

- – **true** et **false** : les booléens.

- – **i** : le nombre imaginaire

- **Les types basiques**

Les booléens

Il s'agit du type **bool**.

Les entiers

Il s'agit du type **int**.

Les réels

Il s'agit du type **real**.

- **Les complexes**

  Il s'agit du type **complex**. A l'affichage, un complexe x+iy est remplacé par le couple (x, y). Quelques fonctions élémentaires associées : **real**, **imag** et **conj**

Les tableaux et les matrices

- Il existe deux types de tableaux, ceux avec des indices entiers, et ceux dont les indices sont des chaines de caractères. Les éléments sont de type **int**, **complex** ou **real**.

On peut aussi préférer des tableaux à deux indices plutôt que le type **matrix**.

real [**int**] a (n) ;

a[3] = 2 ;

Définir une fonction à une variable

```
func   type   nom  fct ( type   var )
{
 instruction 1 ;

 .  . .

  . . .

 instruction   n ;

 return   outvar ;

}
```

Les fonctions dépendant du maillage

- **fespace** espace_name (maillage , type_elements_finis);

Définir un maillage

Maillage triangulaire régulier dans un domaine rectangulaire

- On considère le domaine]x0, x1[x]y0, y1[. Pour générer un maillage régulier nxm, on utilise la commande suivante :

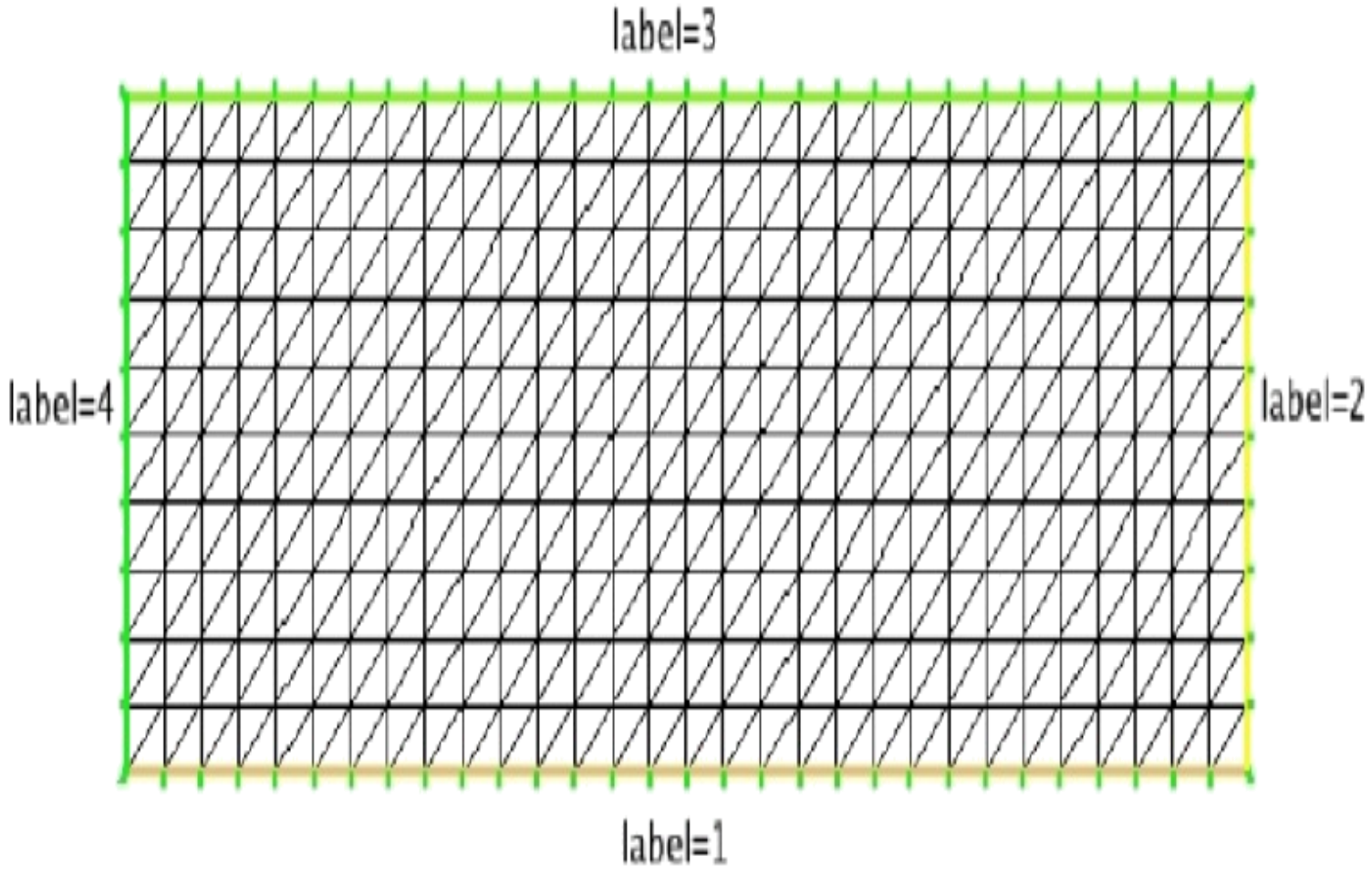
➢ **mesh** nom maillage = **square** (n ,m, [x0+(x1−x0) ∗x , y0+(y1−y0) ∗y]) ;

Maillage triangulaire non structuré défini à partir de ses frontières

- Pour définir les frontières, on utilise la commande **border** :

border name(t=deb , f i n) {x=x (t) ; y=y (t) ; **label**=num label } ;

- On définit ainsi l'ensemble des frontières du domaine. Il faut néanmoins faire attention à l'orientation de la frontière :


label=3
label=4
label=2
label=1

- Pour définir un maillage à partir de ses frontières, on utilise la commande **buildmesh** :
- **buildmesh** nom maillage=**buildmesh** (b1 (z1)+b2 (z2) +...+ bk (zk)) ;
- où les b_i sont les frontières définies avec la commande **border** et z_i un entier relatif dont la valeur absolue représente le nombre de nœuds sur la frontière b_i. Si z_i est négatif, alors l'orientation de la bordure est inversée.

- **savemesh**(nom maillage , nom fichier) ; // permet de sauvegarder le maillage au format .msh

- **readmesh**(nom fichier) ; // permet de l ire un m a i l l a g e à partir d'un fichier .msh

- **mesh** mail2= **movemesh**(mail1 , [f1 (x , y) , f2 (x , y)]) ; /* permet de déformer l e m a i l l a g e mail1 e t de l e s t o c k e r dans mail2 */

- **Mesh** mail2= **adaptmesh**(mail1 , var) /* permet de r a f f i n e r le maillage mail1 dans les zones de fortes variatons de var et de sotcker le Résultat dans mail2 */

Lire les données d'un maillage

Pour accéder à certaines informations du maillage :

- **Th.nt** le nombre de triangles
- **Th.nv** le nombre de noeuds
- **Th[i][j]** le sommet j du triangle i

Ouvrir un fichier

- Pour ouvrir un fichier en lecture :

ifstream name(nom fichier) ;

Pour ouvrir un fichier en écriture :

ofstream name(nom fichier) ;

Résoudre une EDP

Définition de l'espace d'approximation

- On utilise la commande **fespace**.
- **fespace** nom espace (nom maillage , t y p e e l e m e n t s f i n i s) ;

Définir le problème variationnel

- De manière générale, on définit un problème variationnel de la façon suivante :
- **problem** pb name (u , v) = a (u , v)−l (v) + (conditions aux limites) ;
- Pour résoudre un problème variationnel, il suffit de taper la commande :
- pb name ;

Dérivées

- On utilise les commandes **dx** et **dy**, qui ne s'appliquent qu'`a des variables de type éléments finis.
- Exemple :

```
mesh Th=square ( 2 0 , 2 0 , [ x , y]);
fespace Vh(Th, P1) ;
Vh  uh ;
func  u=x+y ;
```

- On peut donc écrire **dx**(uh) mais pas **dx**(u). Si toutefois on a besoin d'éffectuer ce genre d'opération, il faut utiliser la fonction de type éléments finis associée :

```
Vh  ue=u ;
```

Visualiser les résultats

Directement avec Freefem++

On utilise la commande **plot**, qui sert non seulement à afficher des maillages, mais aussi les courbes d'isovaleurs et les champs de vecteurs. Comme pour les commandes **dx** et **dy**, la commande **plot** n'accepte pas de créer les variables de type **func**.

- **plot** (var1 , [var2 , var3] , . . . [l i s t e d ' options]) ;

A travers des exemples

Exemple 1 : (Maillage structuré , condition aux limites homogène)

- Résoudre le problème suivant : trouver u : Ω → R solution de:

$$\begin{cases} -\Delta u(x, y) = f(x, y) & (x, y) \in \Omega, \\ u(x, y) = 0 & (x, y) \in \Gamma, \end{cases}$$

Où Ω =]0, 1[×]0, 1[est le carré unité et Γ = ∂Ω sa frontière.

On choisira comme donnée f(x, y) = xy (par exemple).

Le fichier suivant, nommé laplace1.edp, donne un codage de ce problème en Freefem++

La formulation variationnelle de ce problème consiste à trouver u ∈ V tel que:

$$\int_\Omega \nabla u \, \nabla v = \int_\Omega f \, v \qquad \forall \, v \in V(\Omega) \qquad \text{Avec } V = H_0^1(\Omega)$$

```
//. Fonction f

func f=x*y;

//Définition du maillage

mesh Th=square(20,20);//construit une grille 20*20 du carre unite

plot(Th,wait=1);//affiche le maillage

//espace Eléments Finis P1

fespace Vh(Th,P1); //construit d'après Th

Vh uh,vh; //uh et vh sont des éléments de Vh
```

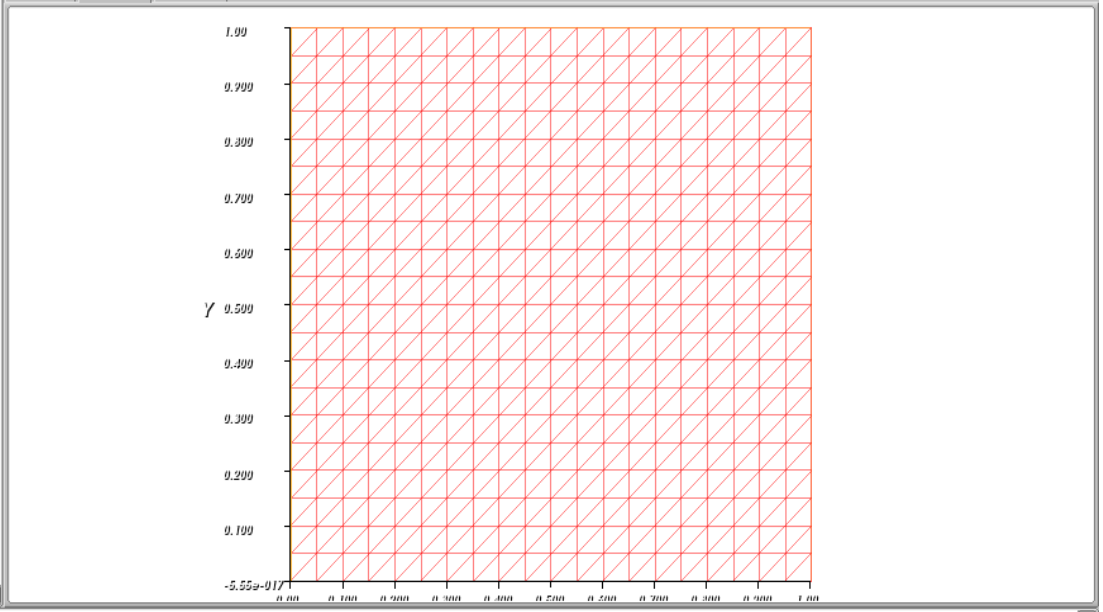
```
//Définition du problème sous forme variationnelle

problem laplace(uh,vh,solver=LU)=

int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))-int2d(Th)(f*vh)+on(1,2,3,4,uh=0);

//NB : le carré à 4 cotes numérotés de 1 a 4
//Résolution du problème

laplace;

//On affiche le résultat

plot(uh,wait=1,fill=true,value=true);

// Exporter vers un fichier texte
{ofstream file("mai.txt");
file<<"2 1 "<<uh[]<<" 2"<<endl;
```

FreeFem++-cs *

File Edit Compute Graphics Options Help

```
1  //. Fonction f
2  func f=x*y;
3  //Définition du maillage
4  mesh Th=square(20,20);//construit une grille 20*20 du c
5  plot(Th,wait=1);//affiche le maillage et attend un clic
6  //espace Eléments Finis P1
7  fespace Vh(Th,P1); //construit d'après Th
8  Vh uh,vh; //uh et vh sont des éléments de Vh
9  //Définition du problème sous forme variationnelle
10 problem laplace(uh,vh,solver=LU)=
11 int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))-int2d(Th)(f*vh)+
12 //NB : le carré à 4 cotes numérotés de 1 a 4
13 //Résolution du problème
14 laplace;
15 //On affiche le résultat
16 plot(uh,wait=1,fill=true,value=true);
17
18
```

version | figure 1 | figure 2




```
 -- Square mesh : nb vertices  =441 ,  nb triangles = 800 ,  nb boundary edges 80
 -- Solve :
        min 2.81111e-064  max 0.0210796
times: compile 0.024s, execution 0.167s,  mpirank:0
 CodeAlloc : nb ptr  2440,  size :163420 mpirank: 0
Bien: On a fini Normalement
```

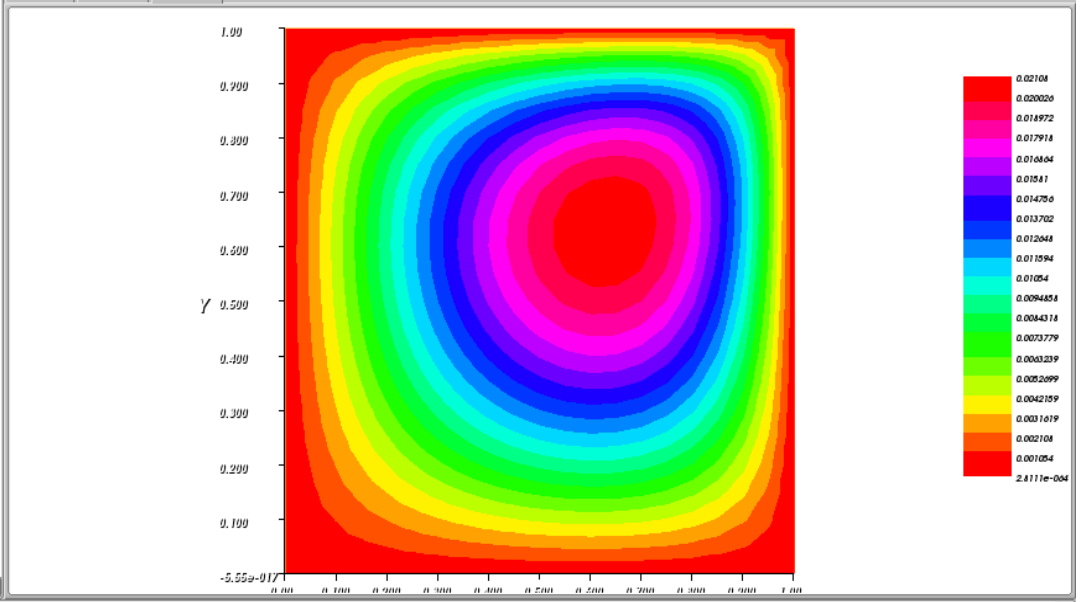
FR 15:33 16/10/2014

FreeFem++-cs *

File Edit Compute Graphics Options Help

```
1  //. Fonction f
2  func f=x*y;
3  //Définition du maillage
4  mesh Th=square(20,20);//construit une grille 20*20 du c
5  plot(Th,wait=1);//affiche le maillage et attend un clic
6  //espace Eléments Finis P1
7  fespace Vh(Th,P1); //construit d'après Th
8  Vh uh,vh; //uh et vh sont des éléments de Vh
9  //Définition du problème sous forme variationnelle
10 problem laplace(uh,vh,solver=LU)=
11 int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))-int2d(Th)(f*vh)+
12 //NB : le carré à 4 cotes numérotés de 1 a 4
13 //Résolution du problème
14 laplace;
15 //On affiche le résultat
16 plot(uh,wait=1,fill=true,value=true);
17
18
```

version | figure 1 | figure 2




```
 -- Square mesh : nb vertices  =441 ,  nb triangles = 800 ,  nb boundary edges 80
 -- Solve :
       min 2.81111e-064  max 0.0210796
times: compile 0.024s, execution 0.167s,  mpirank:0
 CodeAlloc : nb ptr  2440,  size :163420 mpirank: 0
Bien: On a fini Normalement
```

FR 15:36 16/10/2014

Exemple 2 : (Maillage non structuré , condition aux limites homogène)

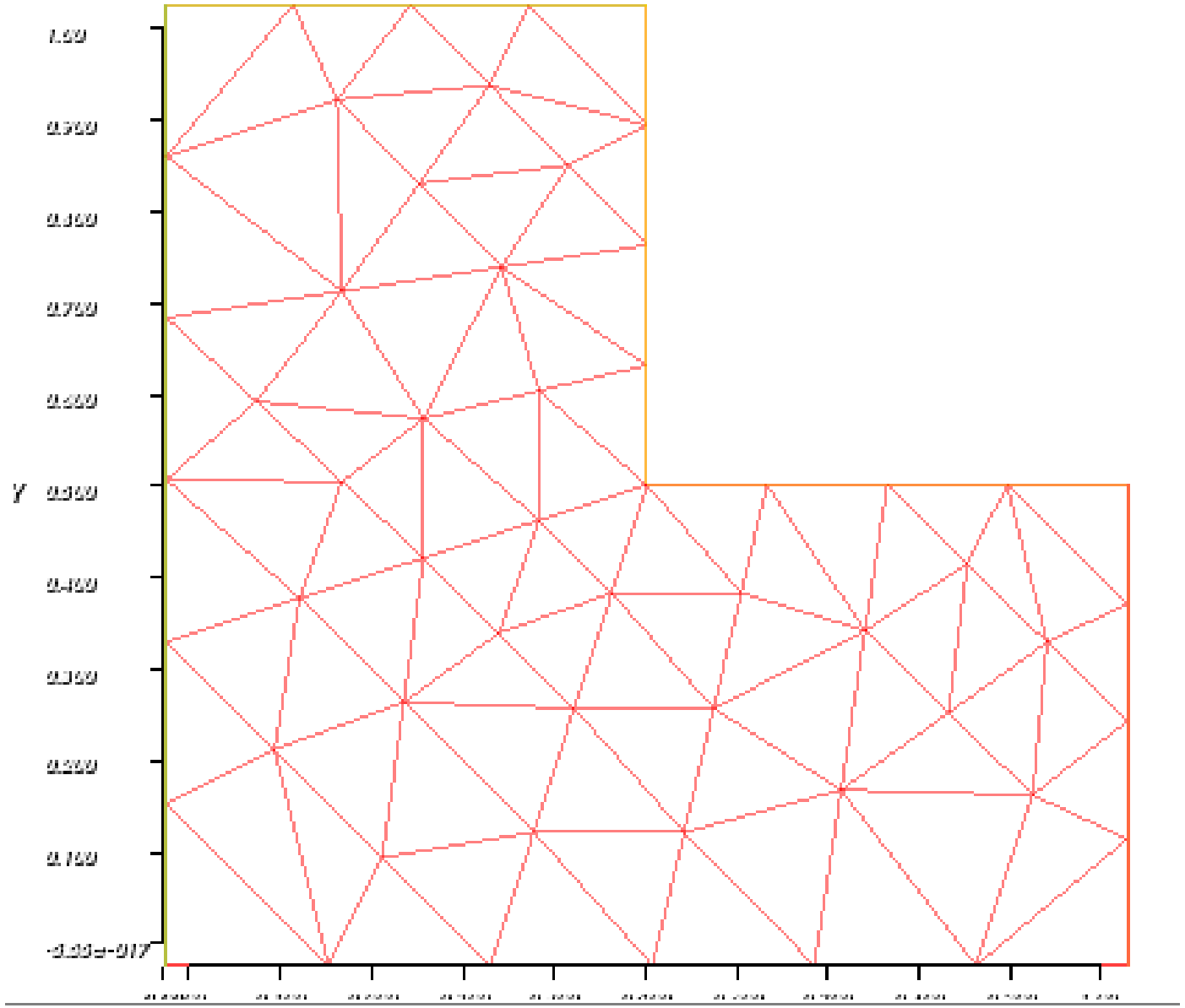
- trouver $u : \Omega \rightarrow R$ solution du problème suivant:

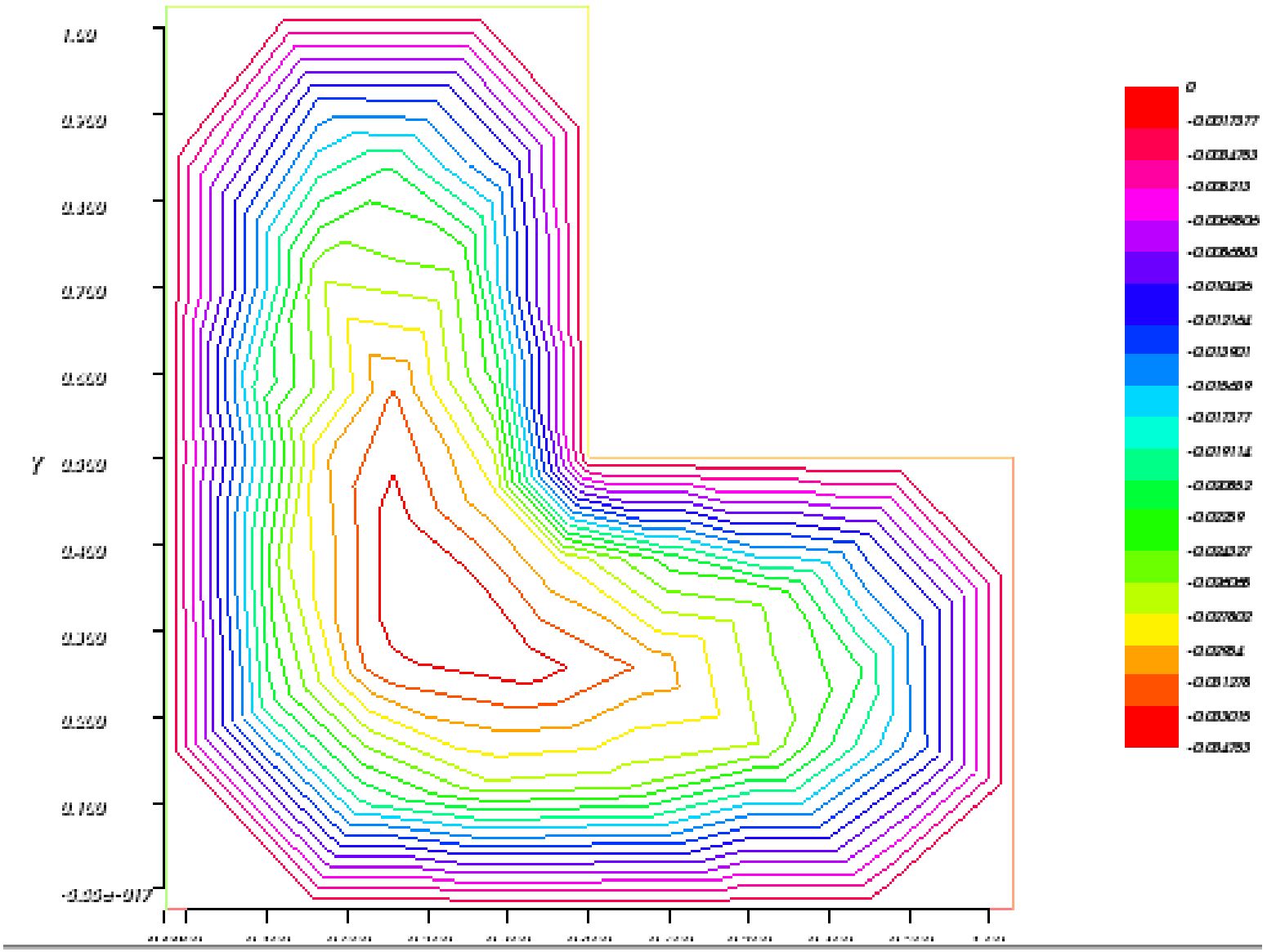
$$-\Delta u(x, y) = f(x, y) \quad (x, y) \in \Omega,$$
$$u(x, y) = 0 \qquad (x, y) \in \Gamma,$$

Où $\Omega =]0, 1[\times]0, 1[$ est le carré unité et $\Gamma = \partial\Omega$ sa frontière.

On choisira comme donnée $f(x, y) = xy$ (

```
//Définition des Maillages
border aaa(t=0,1){x=t;y=0;};
border bbb(t=0,0.5){x=1;y=t;};
border ccc(t=0,0.5){x=1-t;y=0.5;};
border ddd(t=0.5,1){x=0.5;y=t;};
border eee(t=0.5,1){x=1-t;y=1;};
border fff(t=0,1){x=0;y=1-t;};
// Maillage non structuré
mesh Th = buildmesh (aaa(6) + bbb(4) + ccc(4) +ddd(4) + eee(4) + fff(6));
fespace Vh(Th,P1);
Vh uh,vh;
Vh u=0,v;
func g= 1;
func f= x*y;
problem Probem1(uh,vh) =int2d(Th)(  dx(uh)*dx(vh) + dy(uh)*dy(vh)) + int2d(Th) ( f*vh )
  + on(aaa,bbb,ccc,ddd,eee,fff,uh=0)  ;

Probem1;
// plot(u,Th,wait=1);
plot(uh);
plot(Th)
```


1.00
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
-3.33e-017
y

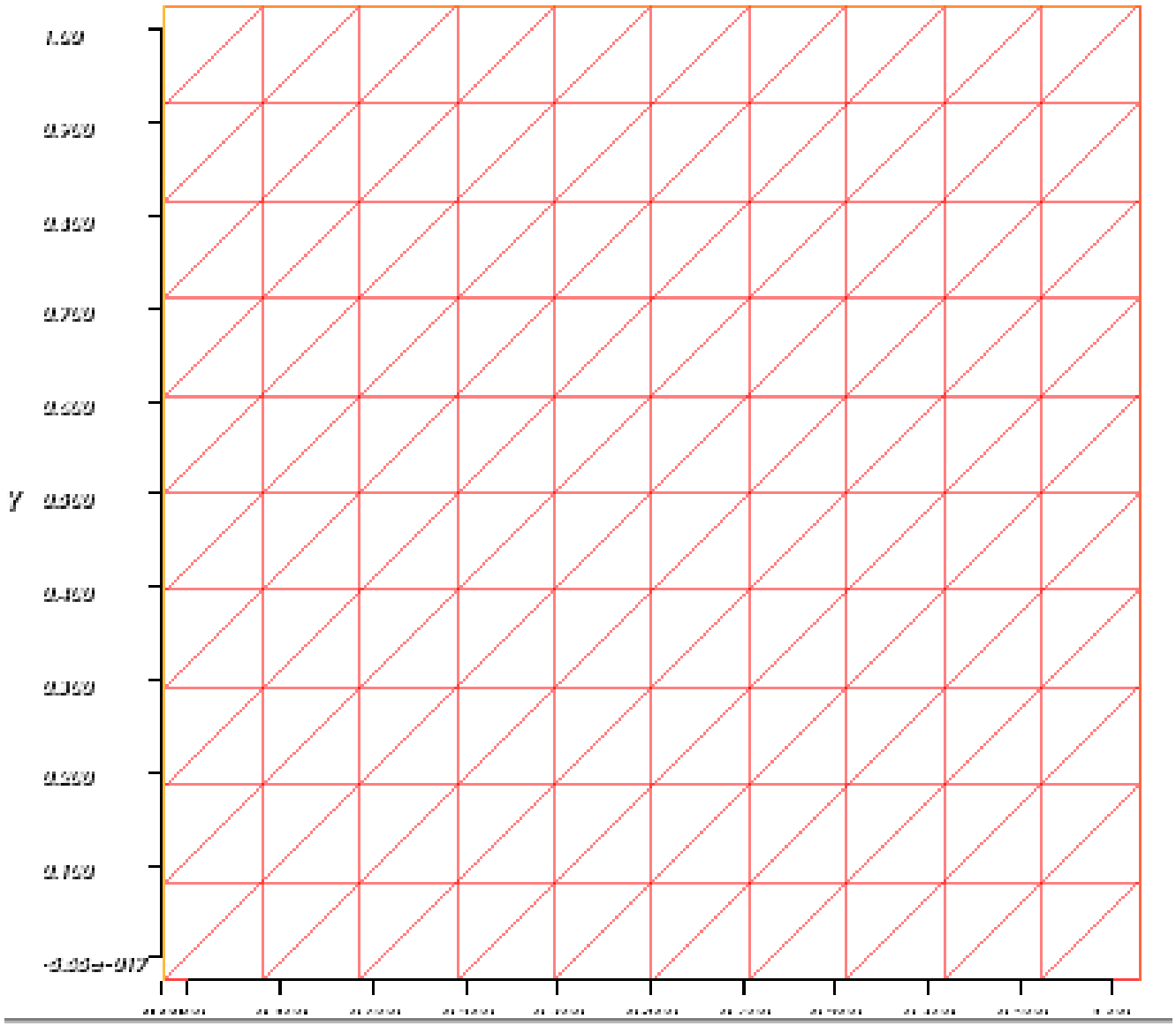

Y
0
-0.0017577
-0.0035153
-0.005273
-0.0070306
-0.0087883
-0.010546
-0.012304
-0.014061
-0.015819
-0.017577
-0.019334
-0.021092
-0.022849
-0.024607
-0.026364
-0.028122
-0.029879
-0.031637
-0.033395
-0.035153

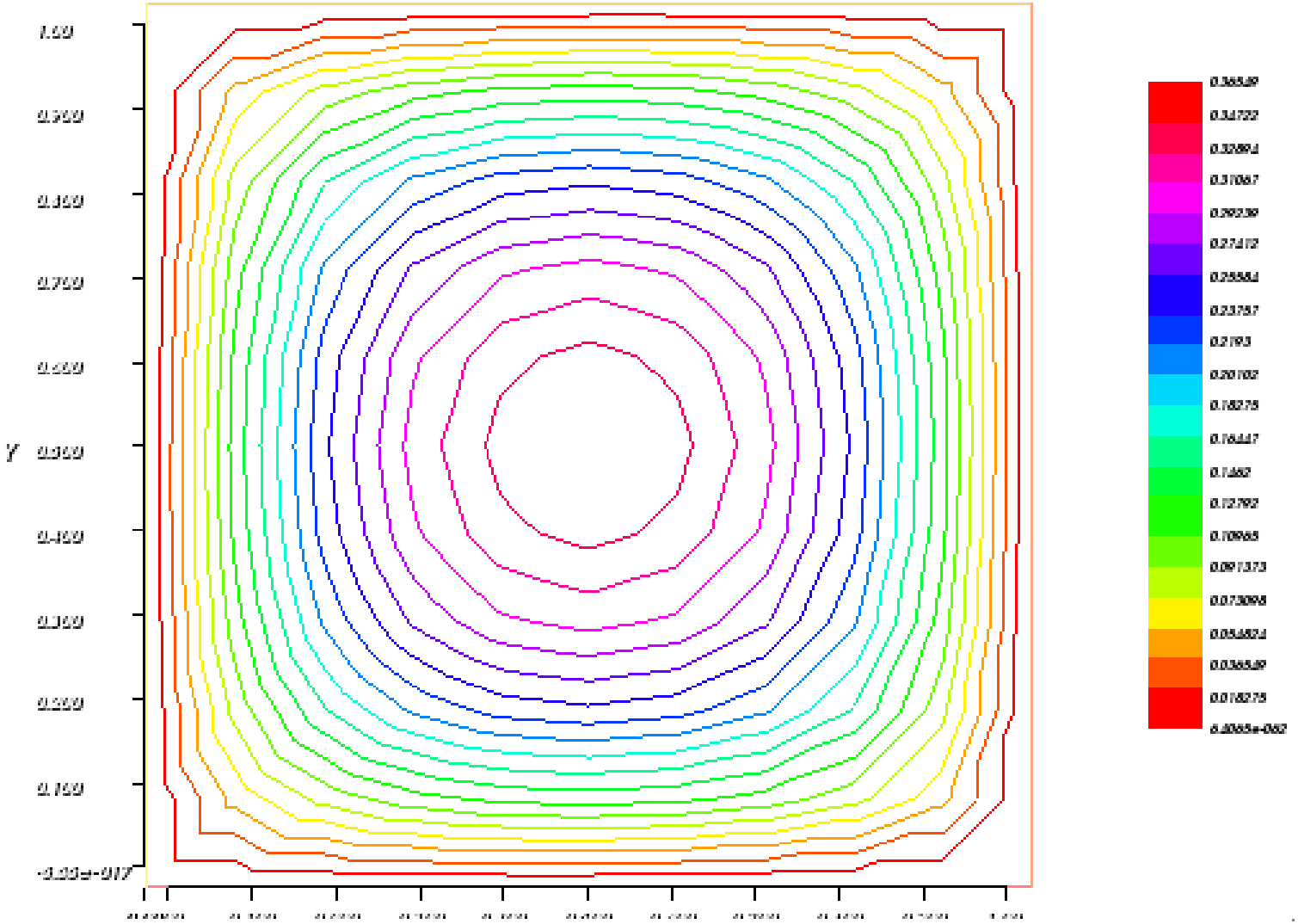
```
//Définir l’éspace du domaine ainsi que le maillage
int Nbnoeuds=10;
mesh Th=square(Nbnoeuds,Nbnoeuds,[x,y]);
fespace Vh(Th,P1) ;

func f=5*sin (2* pi *x ) * sin ( pi *y ) ;
func g=0;

func u=sin (2* pi *x ) * sin ( pi*y ) ;
Vh uh , vh , ue ;

 // Définition du problème variationnel sur Th
problem P(uh , vh ) = int2d (Th) (dx(uh) *dx( vh )+dy(uh) *dy( vh ) )
- int2d (Th) ( f *vh )+ on( 1 , 2 , 3 , 4 , uh=0) ;
// Résoltion du problème
P;
// Affichage de la solution
plot (uh , ps="lapdirichlet.ps");
```

```
int np1=0;
//Compte le nombre de points de quadrature à la frontière 2
int1d(Th,2,qfe=qf1pElump)( real(np1++));

real[int] xx1(np1),yy1(np1);

{
int i=0,j=0;
int1d(Th,2,qfe=qf1pElump)((xx1[i++]=x)+(yy1[j++]=y));
}
cout << " xx1 = " << xx1 << endl;
cout << " yy1 = " << yy1 << endl;
plot(Th);
plot(uh);
// Exporter vers un fichier texte
 {ofstream file("mai.txt");
file<<"2 1 "<<uh[]<<" 2"<<endl;
}
```

```
//définir la même formulation variationnelle mais cette fois avec le type varf

int Nbnoeuds=10;
mesh Th=square(Nbnoeuds,Nbnoeuds,[x,y]);

// Dé f in i t i o n de l'espac e  d ' approximation
fespace Vh(Th,P1) ;

func f=5*sin (2* pi *x ) * sin ( pi *y ) ;
func g=0;

func u=sin (2* pi *x ) * sin ( pi*y ) ;
Vh uh , vh , ue ;

varf PA(uh,vh)=int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh));

//définir la matrice associée à la forme variationnelle
matrix A=PA(Vh,Vh);
 cout << "les noeuds" <<Th.nv << endl;
cout << "lA MATRICE A est" <<A  << endl;
```

```
varf PB(uh,vh)=int2d(Th)(f*vh)+on(1,2,3,4,uh=g);
real[int] B=PB(0,Vh);
  cout << "le vecteur B est" <<B  << endl;
uh[]=A^-1*B;
plot(uh);
 cout << "la solution est " << uh[] << endl;
int np1=0;
int1d(Th,2,qfe=qf1pElump)( real(np1++));
 real[int] xx1(np1),yy1(np1);
 {
int i=0,j=0;
int1d(Th,2,qfe=qf1pElump)((xx1[i++]=x)+(yy1[j++]=y));
}
cout << " xx1 = " << xx1 << endl;
cout << " yy1 = " << yy1 << endl;
plot(Th);
plot(uh);
 {ofstream file("mai.txt");
file<<"2 1 "<<uh[]<<" 2"<<endl;
}
```


1.00
0.900
0.800
0.700
0.600
y 0.500
0.400
0.300
0.200
0.100
-3.33e-017


Y
0.365
0.347
0.329
0.310
0.292
0.274
0.255
0.237
0.219
0.201
0.183
0.164
0.146
0.128
0.109
0.0913
0.0730
0.0548
0.0365
0.0183
6.4065e-002

Exemple 3 : (Maillage non structuré , condition aux limites non-homogène)

- On considère le problème suivant:

$$\begin{cases} -\Delta u = f & dans\ \Omega \\ u = xy & sur\ \partial\Omega \end{cases}$$

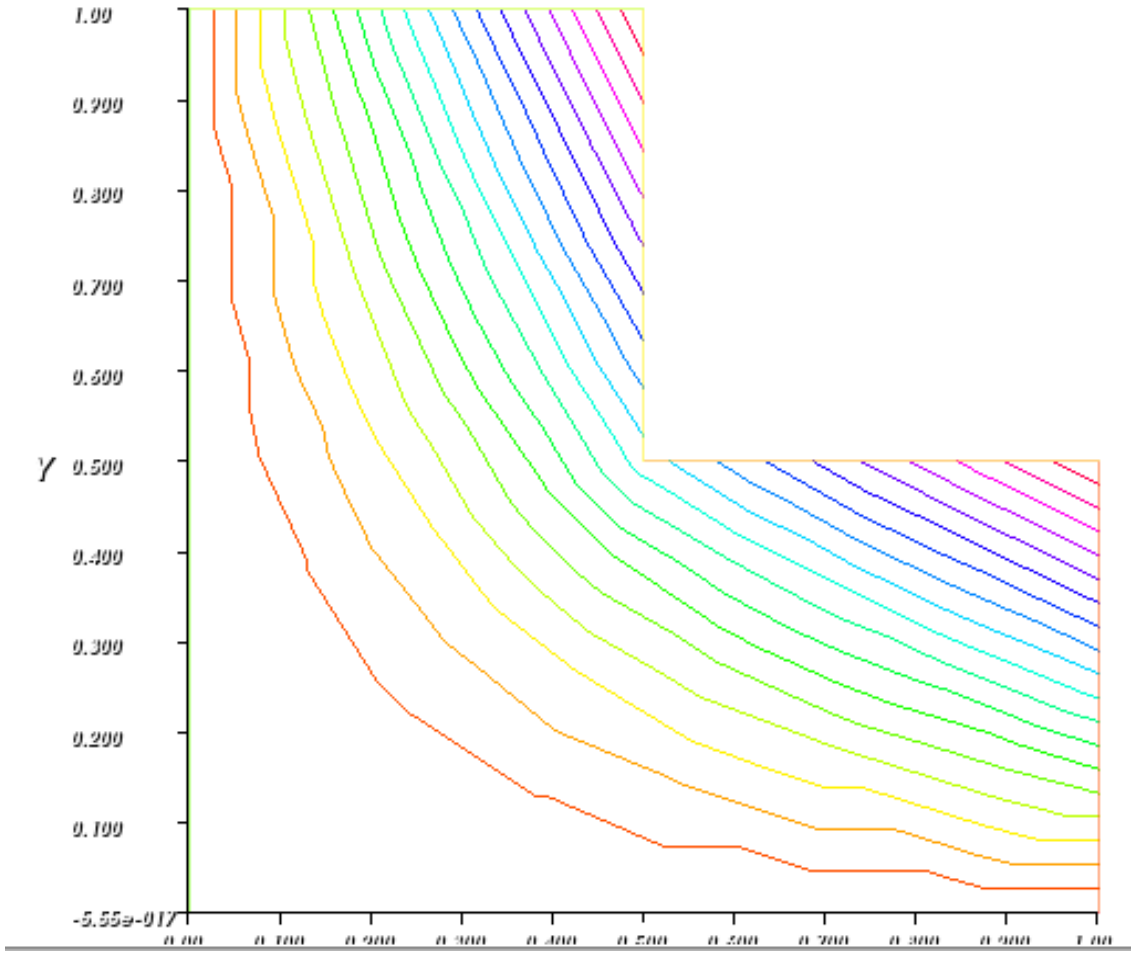
Le domaine Ω est : $\Omega =]0,1[^2 \setminus [\frac{1}{2},1]^2$

et la fonction f de sorte que l'on connaisse la solution exacte du problème afin de calculer l'erreur .

On choisira donc f = $5\pi^2$ sin 2πx sinπy de sorte que la solution est:

u = sin 2πx sin πy.

```
//Définition des Maillages
border aaa(t=0,1){x=t;y=0;};
border bbb(t=0,0.5){x=1;y=t;};
border ccc(t=0,0.5){x=1-t;y=0.5;};
border ddd(t=0.5,1){x=0.5;y=t;};
border eee(t=0.5,1){x=1-t;y=1;};
border fff(t=0,1){x=0;y=1-t;};
// Maillage non structuré
mesh Th = buildmesh (aaa(6) + bbb(4) + ccc(4) +ddd(4) + eee(4) + fff(6));
fespace Vh(Th,P1;
Vh u=0,v;
func f= 1;
func g= x*y;

problem Probem1(u,v) =
   int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v))
  + int2d(Th) ( v*f )
  + on(aaa,bbb,ccc,ddd,eee,fff,u=g)  ;

Probem1;
// plot(u,Th,wait=1);
plot(u);
plot(Th);
```


1.00
0.900
0.800
0.700
0.600
Y
0.500
0.400
0.300
0.200
0.100
-5.55e-017

Exporter vers un fichier texte

Le fichier exemple.txt Voilà le code permettant de générer ce fichier :

```
{ ofstream file("exemple.txt");
  file<<"2 1 "<<uh[].n<<endl;
• For (int j=0; j<uh[].n; j++) {
file <<uh[][j]<<endl; } }
```

Conditions aux limites de Dirichlet

On utilise la commande **on** sous la forme : **on**(num1 , numk , u=g) ;

- On peut définir une formulation variationnelle avec le type **varf**, et ce de la même manière que le type **problem**. Cela présente un grand intérêt dans le cas où l'on veut résoudre plusieurs problèmes présentant une partie commune. Le deuxième intérêt est de pouvoir raisonner en terme de produits matriciels. Pour définir la matrice associée à une forme variationnelle, on procède comme suit :

matrix name = varf name (espace1 , espace2) ;