

Méthodes Numériques pour les Equations aux Dérivées Partielles

Olivier Pironneau

November 26, 1997

Contents

0.1	Préface	3
0.2	Dates	4
0.3	Qu'apporte le web?	4
0.4	Diplome	4
0.5	Qui corrige les exercices?	5
0.6	Le programme	5
0.7	Que faire pour se préparer	6
1	LEÇON: Introduction au calcul scientifique	6
1.1	Calcul scientifique: définition	6
1.2	Un exemple simple	6
1.3	Discrétisation	7
1.4	Algorithme	7
1.5	Programmation	8
1.6	Les Exercices de la leçon	11
2	LEÇON freefem et gnuplot	16
2.1	Orientation	16
2.2	Le problème	16
2.3	Un exemple pour freefem	17
2.4	Fichier triangulation	18
2.5	Relecture du fichier triangulation	19
2.6	Tracé avec gnuplot	21
2.7	Visualisation d'une fonction	23
2.8	Complément: le debugger gdb	24
2.8.1	La commande where	24

2.8.2	Passage de paramètre sous unix	25
2.9	Tracé d'une fonction de 2 variables	26
2.9.1	Principe: Interpolation P1	26
2.9.2	Estimations d'erreur	28
3	LEÇON Les classes Triangulation	28
3.1	Première amélioration	28
3.2	Protection contre les dépassements de tableau	29
3.3	Deuxième amélioration	31
3.4	Construction du tableau des arêtes	33
3.5	Les Exercices de la leçon	34
4	LEÇON Un programme d'Elements Finis P1	37
4.1	Introduction	37
4.2	Algorithme du gradient conjugué	37
4.3	Structures de données	38
4.3.1	La Classe Vector	38
4.3.2	Fonctions P1	40
4.3.3	La classe Problem	42
4.3.4	Remarque sur l'assemblage	44
4.4	Les Exercices de la leçon	45
4.5	Directives pour cet exercice	47
4.5.1	Dans P1::intgrad2	47
4.5.2	Changement dans Laplace::derivener	47
4.5.3	Modification dans Laplace::solvgadconj	48
4.5.4	Remarque sur Ikl	49
4.5.5	Changement dans Laplace::Laplace	49
4.5.6	Gén/ération de données	49
4.5.7	Vérification	50
5	LEÇON Triangulation Automatique	51
5.1	Triangulation automatique d'un rectangle.	51
5.1.1	Le problème:	51
5.1.2	Algorithme	51
5.1.3	Le critère de Delaunay	52
5.1.4	Algorithme	52
5.2	Triangulation d'un domaine convexe	52
5.2.1	Algorithme	53
5.3	Triangulation d'un domaine non convexe	53
5.3.1	Algorithme	53

5.3.2	Comment donner les points internes	54
5.4	Un premier module de triangulation automatique en C	54
6	LEÇON : Frontières courbes, triangulation automatique et adaptative	57
6.1	Un nouveau mailleur	57
6.1.1	Fichier frontière	57
6.1.2	Mots clefs du fichier frontière	59
6.1.3	Conversion du premier vers le second trianguleur	60
6.1.4	L'algorithme employé	61
6.2	Compléments	62
6.2.1	Polygones de Voronoi	62
6.2.2	Quadtrees	62
6.3	Maillage adaptatif	63
6.3.1	Fichier pour la triangulation et le maillage adaptatif	65
6.4	Renumerotations	65
6.4.1	Une fonction de renumérotation	70
6.4.2	lecture de chaîne de caractères dans DBI	73
6.4.3	Bug dans DBI	73
7	Projet	74
7.1	Orientation	74
7.2	L'objectif	74
7.3	Les étapes	75
7.4	Aide à la réalisation du projet	77
7.4.1	L'instruction "save"	77
7.4.2	L'instruction "solve"	77
7.4.3	L'instruction "adaptmesh"	78
7.4.4	Casting de u au format DBI	79
8	LEÇON: L'interpreteur DBI	81

0.1 Préface

Le cours de la Maîtrise d'ingénierie mathématique de Paris 6, intitulé

"INTRODUCTION AU CALCUL SCIENTIFIQUE EN C++"

de *Dominique Bernardi* et *Olivier Pironneau* sera sur le réseau www à partir du 10 Février 1997. Il s'agit d'une nouvelle expérience de télé-enseignement pour un cours bien rodé et dont le support papier existe pour

la partie théorique et pour l'apprentissage du C++ sous la forme de deux livres :

- - introduction au calcul scientifique de B. Lucquin et O. Pironneau (Masson 96)
- - G. Buzzi-Ferrari: Scientific C++ (Addison-Wesley 1993)

Le cours se présente sous la forme d'une session par semaine avec des exercices. Un exercice est **obligatoire** et la solution (un programme en général) doit être renvoyé au professeur.

REMARQUE 1

Ce fichier ayant été préparé avec le logiciel latex2html, il est lisible directement:

- ramener sur votre machine le fichier "infosci.tex" (il se trouve dans le site ftp de Pironneau (<ftp://ftp.ann.jussieu.fr/pub/soft/pironneau/infosci.tex>)).
- compiler le fichier avec latex2e. (remplacer toute ligne commençant par \html par %\html)

0.2 Dates

Attention ce cours est en cours d'écriture. Compter une leçon par semaine à partir du 10 Février.

0.3 Qu'apporte le web?

Par rapport à un livre, il permet

- le dialogue avec l'auteur,
- la correction des exercices,
- de l'hypertexte.

0.4 Diplome

Il n'est pas possible actuellement de faire valider ce cours si vous n'êtes pas inscrit à Paris 6. Une idée serait de pouvoir faire valider ce cours dans d'autres maîtrises à condition d'être contrôlé par un enseignant de la maîtrise locale avec lequel un des auteurs pourrait dialoguer par email. Cela demande, bien sûr, l'autorisation du responsable de la maîtrise locale.

0.5 Qui corrige les exercices?

D.Bernardi, O. Pironneau et C. Prud'homme ne pourront pas encadrer plus de 30 étudiants sur réseau: dès maintenant.

0.6 Le programme

Le programme de l'UV est :

- Le langage C++ (*)
- Introduction à la théorie des langages informatiques
- Introduction à l'informatique graphique
- Les commande de base d'UNIX (*)
- Quelques fonctions C de Xwindows (*)
-
- Programmation de la méthode des éléments finis
- Stockage Morse, méthodes itératives et méthodes directes pour $Ax=b$
- Les EDP linéaires de la physique
- La méthode des différences finis pour les problèmes d'évolution
- La méthode de GMRES pour les problèmes non-symétriques

Ce cours est enseigné à Paris 6 (10 Février - 10 Juin environ) en 2x2h de cours et 8h de TD par semaine (mais on ne demande pas de travail chez soi en plus); évidemment sur le web tout le travail est chez soi, c'est à dire probablement 12h/semaine. C'est pourquoi les parties avec (*) ne seront pas sur le web. En particulier le C++ sera appris "sur le tas". En plus des exercices, le contrôle des connaissances se fait par l'écriture d'un projet genre freefem Cette année le projet sera l'inclusion de la différentiation automatique dans freefem.

0.7 Que faire pour se préparer

- 1. Avoir accès à une machine (UNIX de préférence) branchée sur le WEB. Si vous disposez d'un PC musclé vous pouvez essayer d'installer linux voir aussi les autres sites. Sinon vous pouvez travailler sur un PC Windows 95 + Visual C++ ou un Mac + Metrowerks C++ mais vous ne pourrez pas faire tous les exercices.
- 2. Vérifier votre compilateur C++, de préférence gcc 2.7 g++ (c'est gratuit) sous X. Il est aussi assez important d'avoir un "débugger" (gdb 4.16 avec gcc), ainsi qu'un éditeur de texte (vi sous unix ou mieux xemacs)
- 3. Télécharger et exécuter un exemple de freefem
- 4. Télécharger et exécuter un exemple de gnuplot 3.5
- 5. Commander les 2 livres (ok Pironneau recoit 4F par livre commandé, mais il est prêt à les reverser aux lecteurs choqués).

1 LEÇON: Introduction au calcul scientifique

1.1 Calcul scientifique: définition

La terminologie *calcul scientifique* désigne tout calcul à l'usage de la science ; en fait nous nous limiterons dans cet ouvrage à des applications issues de la physique ou de l'ingénierie.

L'ingénierie couvre les domaines de la mécanique des fluides ou des structures, l'électromagnétisme, et bien d'autres domaines encore ; la liste de toutes les applications du calcul scientifique est bien trop longue pour pouvoir être détaillée ici.

1.2 Un exemple simple

Nous souhaitons connaître la température T d'une barre métallique dont les deux extrémités sont maintenues aux températures T_0, T_M et plongée dans une atmosphère ambiante elle-même à une température donnée T_e . Cette barre est assimilée à un segment de droite $[0, L]$.

Dans ce problème, il y a une perte de la chaleur due à la convection de l'air, que l'on peut modéliser par une fonction $a(x)$; la température T est alors solution de l'équation différentielle ordinaire

$$-k \frac{d^2 T}{dx^2} + a(x)(T - T_e) = 0, \quad 0 < x < L,$$

où k désigne un coefficient de diffusion thermique. A cette équation s'ajoutent les conditions aux limites

$$T(0) = T_0, \quad T(L) = T_M$$

Les mathématiciens nous disent que si k est strictement positif et si a est une fonction à valeurs positives, alors le problème est bien posé, signifiant par là qu'il admet en particulier une solution et une seule. Ceci est une information importante à double titre : nous pouvons envisager un calcul (approché) de cette solution, et si le programme que nous écrivons ne marche pas, la faute incombera nécessairement au programmeur!

1.3 Discrétisation

Nous commençons par discrétiser l'équation différentielle par *différences finies*. Cette méthode, que nous expliquerons plus longuement ultérieurement, consiste à approcher les termes de dérivation par des quotients différentiels, se basant sur la définition de la dérivée qui permet d'écrire, que pour h petit, on a :

$$\frac{df}{dx}(x) \simeq \frac{f(x+h) - f(x)}{h}.$$

Les termes d'ordre 2 sont approchés par :

$$\frac{d^2f}{dx^2}(x) \simeq \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

L'intervalle $[0,L]$ est alors décomposé en M intervalles de longueur $h=L/M$, et la solution est recherchée en chacun des points de subdivision $x_m = m h$, $m = 0, \dots, M$; notons T_m cette valeur.

Le problème approché est équivalent à la résolution du système linéaire suivant (T_0 et T_M donnés)

$$-\frac{k}{h^2}[T_{m+1} - 2T_m + T_{m-1}] + a_m(T_m - T_0) = 0, \quad m = 1, \dots, M-1.$$

La théorie nous dit que plus le maillage est fin (ie. plus h est petit), meilleure est l'approximation : l'erreur $T_m - T(x_m)$ tend vers 0 quand h tend vers 0.

1.4 Algorithme

Pour résoudre le système linéaire $Ax = b$, la méthode de Gauss-Seidel consiste à définir par récurrence une suite de valeurs pour $x^{(j)}$, convergeant vers x , en résolvant

$$(D - E)x^{(j+1)} = Fx^{(j)} + b,$$

où nous avons utilisé la décomposition suivante de A : $A=D-E-F$, avec D matrice diagonale dont les éléments diagonaux sont ceux de A, -E matrice triangulaire inférieure stricte, dont les éléments non nuls sont ceux de la partie triangulaire inférieure stricte de A et $-F=A-D+E$ partie triangulaire supérieure stricte de A. La théorie nous dit que si la matrice A est symétrique définie positive (ce qui est précisément le cas ici), cette méthode est convergente, dans le sens où quand le nombre d'itérations j devient très grand, l'erreur entre la solution exacte x et la solution approchée tend vers 0.

La programmation de cet algorithme pour la résolution du système consiste à extraire la valeur de T_m de l'équation m, ce qui donne :

$$\forall m \in \{1, \dots, M-1\}, \quad T_m^{(j+1)} = \left\{ \frac{k}{h^2} [T^{(j)}_{m+1} + T^{(j+1)}_{m-1}] + a_m T_e \right\} / \left[a_m + 2 \frac{k}{h^2} \right],$$

T_0 et T_M étant donnés.

Remarquons que l'indice j de l'itération de Gauss-Seidel n'est pas nécessaire dans la programmation qui peut se faire en rangeant $T^{(j+1)}, T^j$ dans la même mémoire. En effet, il n'est pas nécessaire de stocker en mémoire les vecteurs correspondant à 2 itérations successives. Au fur et à mesure du calcul (ie. quand m augmente) la valeur de T_m à l'itération j est remplacée par sa nouvelle valeur à l'itération j+1.

Cet algorithme est itéré autant de fois que nécessaire, ie. jusqu'à ce que les valeurs de T_m à deux étapes successives soient presque les mêmes.

1.5 Programmation

Le programme en C++ est le suivant:

```
#include <stdlib.h>
#include <math.h>
#include <iostream.h>
#include "rgrafpor.h"

const int M = 100;
const float dx = 0.1;
const int jmax = 200;

typedef enum{ texte, graphique } affiche;

class data
```

```

{ public:
  float k, T0, TM, Te;
  float a[M];
  void read();
};

void init( float* T, data& d);
void relax( float* T, data& d);
void showresult( affiche mode, float* T);

void data::read()
{
cout << "enter k:";   cin >> k;
cout << "enter Te:";  cin >> Te;
cout << "enter T0:";  cin >> T0;
cout << "enter TM:";  cin >> TM;
for(int m = 0; m< M ; m++)
  a[m] = float(m)*float(m)/(M-1.)/(M-1.);
}

void init( float* T, data& d)
{
for(int m = 0; m<= M; m++)
  T[m]=d.T0 * float(M- m)/float(M) + d.TM *float(m)/float(M);
}

void relax( float* T, data& d)
{
  const float kdx2 = d.k / dx / dx;
  for(int m = 1; m< M; m++)
    T[m] = (kdx2 * (T[m+1] +T[m-1])
    + d.a[m] * d.Te) / (d.a[m]+ 2 * kdx2);
}

void showresult( affiche mode, float* T)
{ int m;
  if(mode==graphique)
  {
    initgraphique();
    move2( 0, (int)T[0]);
  }
}

```

```

        for( m = 1; m<= M; m++)
            line2( m, (int)T[m]);
        rattente(1);
        closegraphique();
    } else if(mode==texte)
        for( m = 0; m<= M; m++)
            cout<<T[m]<<endl;
}

void main()
{
float* T = new float[M+1];
data d;

d.read();
init(T, d);
    for (int j= 0; j< jmax; j++)
        relax(T, d);

showresult(texte, T);
}

```

Cet exemple simple nous montre les quatre étapes d'un logiciel de calcul scientifique :

- mise en équation, choix de la discrétisation et de l'algorithme
- entrée des données (pré-processeur)
- calcul
- visualisation (post-processeur).

Les étapes de programmation sont les étapes 2 à 4, l'étape 1 étant une étape de modélisation et d'analyse numérique. Même sur cet exemple simple, le nombre d'instructions dans les parties pré et post-processeurs est plus grand que le nombre d'instructions dans la partie calcul. Cela signifie que le programmeur a passé plus de temps à écrire les pré et post-processeurs. Cette situation est assez générale en calcul scientifique, et c'est la raison principale pour laquelle les personnes qui programment préfèrent utiliser

des logiciels commerciaux plutôt que d'écrire eux-mêmes leurs propres programmes d'entrée des données ou d'exploitation graphique.

Enfin remarquons que le programme que nous avons écrit n'est pas très *convivial*, et ceci pour plusieurs raisons :

- L'entrée des données doit se faire dans un ordre bien précis et des messages d'erreurs doivent permettre d'en refaire éventuellement l'exécution. A l'heure actuelle, les programmes les plus conviviaux ont des menus, et l'utilisateur indique au programme (avec la souris) quelle donnée doit être entrée en premier.
- La façon dont les données sont entrées n'est pas optimale. Supposons en effet que nous changions $M = 10$ en $M = 100$ pour augmenter la précision. L'utilisateur aura alors besoin d'entrer à la main 100 valeurs pour définir le tableau a . Pour éviter cette besogne fastidieuse, il est préférable de lire les valeurs de ce tableau dans un fichier, ou bien de les définir à partir d'une fonction, par exemple en écrivant:

$$a=x*x-1,$$

et c'est le pré-processeur qui génère ensuite le tableau a .

- Enfin la partie post-processeur est vraiment rudimentaire. Il n'y a aucune mise à échelle de la fonction, de sorte que si certaines valeurs qu'elle prend sont trop importantes, une partie de sa représentation graphique sera en dehors de l'écran. Par ailleurs, aucun axe n'est tracé, et la figure ne comporte pas de légende...

1.6 Les Exercices de la leçon

EXERCICE 1

- 1. Executer le programme précédent dans un environnement UNIX. Pour ce faire on commencera par le mode texte. En principe les autres instructions graphiques n'étant pas appelées le programme s'exécutera quand même, malgré le message de l'éditeur de liens signalant l'absence de code pour `initgraphique()`, `closegraphique()`, `move2()`, `line2()`.
tapera les commandes UNIX suivantes (si votre compilateur n'est pas gcc changer g++)

```
g++ heatbeam.cpp
a.out
```

- 2. Essayez d'obtenir un graphique du résultat en linkant avec le fichier xrgraph.cpp et rgraph.h (voir sources ci-dessous; attention l'axe des "y" pointe vers le bas).
- 3. translatez le dessin, adapter les échelles et rajouter le tracé des axes.

EXERCICE 2 (*Obligatoire*)

- 1. Remplacer la fonction relax par une résolution par la méthode de résolution d'un système tridiagonal en factorisation LU. On ne stockera que les 2 diagonales de L la 2eme diagonale de U (l'autre valant 1) (voir livre chapitre 3, par 2.3 [3]).
- 2. Pour éliminer les fonctions globales init(), showresult() et relax(), introduire une classe temperature ayant ces fonctions comme méthode et réécrire le programme.

REMARQUE 2

Pour obtenir une copie de ces fichiers vous pouvez, soit copier coller à partir du "browser" (Netscape...) soit ramener le fichier latex "infosci.tex" dans le même sous répertoire public-html de pironnea.

fichier makefile pour gcc

```
.SUFFIXES: .cpp
# make variables
export CXX = g++
export CXXFLAGS = -g
export CXXINC = -I/usr/X11/include
export CXXLIBS = -L/usr/X11/lib -lX11 -lm
### sources
OBJS = heatbeam.o rgrafpor.o
heat :$(OBJS)
$(CXX) -o $@ $(OBJS) $(CXXLIBS)
.cpp.o:
$(CXX) $(CXXFLAGS) $(CXXINC) -c $<
.PHONY: clean
```

```

###$
clean:
rm -f *.o *~ varia core
### dependencies
rgrafpor.o: rgrafpor.h
heatbeam.o:rgrafpor.h
### end makefile

// fichier rgrafpor.h
//=====
void initgraphique();
void closegraphique();
void rattente(int waitm);
void move2(int x, int y);
void line2(int x, int y);
void reffecran();

// fichier rgrafpor.cpp
//=====
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "rgrafpor.h"

static Display *display;
static Window win;
static XSizeHints size_hints;
static GC gc;
static XFontStruct *font_info;
static int screen, width, height, currx, curry;

void closegraphique()
{
    XUnloadFont(display, font_info->fid);
    XFreeGC(display, gc);
    XCloseDisplay(display);
}

```

```

void rattente(int waitm)
{
    char click[] = "Click to continue...";
    char values[256];
    XEvent report;

    if (waitm)
    {
        XDrawString (display,
                    win,
                    gc,
                    5,20,
                    click,
                    strlen(click));
        do XNextEvent(display, &report);
        while (report.type != ButtonPress && report.type != KeyPress);
    }
    XCheckMaskEvent(display, ButtonPressMask,&report);
    if (report.type == ButtonPress)
    XFlush (display);
}

int xerror()
{
    fprintf(stderr, "Probleme avec X-Windows\n");
    return 1;
}

void initgraphique()
{
    XEvent report;
    display = XOpenDisplay(NULL);
    font_info = XLoadQueryFont(display, "7x13");
    XSetErrorHandler((XErrorHandler)xerror);
    XSetIOErrorHandler((XIOErrorHandler)xerror);
    screen = DefaultScreen(display);
    width = DisplayWidth(display, screen) - 100;
    height = DisplayHeight(display, screen) - 160;
    win = XCreateSimpleWindow(display, RootWindow(display, screen),

```

```

        50, 80, width, height, 4, BlackPixel(display, screen),
        WhitePixel(display, screen));

size_hints.flags = PPosition | PSize;
size_hints.x = 0;
size_hints.y = 0;
size_hints.width = width;
size_hints.height = height;

XSetStandardProperties(display, win, "plot", NULL, 0, NULL, 0, &size_hints);
XSelectInput(display, win, ExposureMask | ButtonPressMask);
gc = XCreateGC(display, win, 0, NULL);
XSetFont(display, gc, font_info->fid);
XSetForeground(display, gc, BlackPixel(display, screen));
XMapWindow(display, win);
do XNextEvent(display, &report); while (report.type != Expose);
}

void move2(int x, int y)
{
    currx = x;
    curry = y;
}

void line2(int x, int y)
{
    int newx = x, newy = y;
    XDrawLine(display, win, gc, currx, curry, newx, newy);
    currx = newx;
    curry = newy;
}

void reffecran()
{
    XClearWindow(display, win);
}

```

REMARQUE 3

Voici un makefile pour le compilateur x1C de IBM qui suppose qu'on rebatise les fichiers .cpp en .C.

```
heat: heat.o rgrafpor.o
xlC -o heat heat.o rgrafpor.o -lX11 -lm
```

```
rgrafpor.o: rgrafpor.C rgrafpor.h
xlC -c rgrafpor.C
```

```
heat.o: heat.C heat.h rgrafpor.h
xlC -c heat.C
```

2 LEÇON freefem et gnuplot

2.1 Orientation

Le logiciel freefem a été conçu pour faciliter la préparation des données et la visualisation des résultats lors de la résolution numérique d'équations aux dérivées partielles.

En fait freefem va plus loin: c'est un exemple de réalisation possible une fois que les techniques expliquées dans ce cours seront maîtrisées. C'est donc un outils pour la simulation numérique d'EDP. On pourra l'utiliser aussi pour vérifier un calcul fait par un autre code, par exemple. Ici nous

allons d'abord apprendre, sur un exemple simple, les rudiments du langage Gfem qui est au coeur de freefem , puis nous allons montrer comment on peut améliorer les graphiques en utilisant un autre logiciel: gnuplot .

2.2 Le problème

Soit à visualiser la fonction

$$f(x, y) = xy, \quad \forall \{x, y\} : x^2 + y^2 < 1.$$

Une fonction n'est pas seulement définie par ses *valeurs*, mais aussi par son *domaine*. Dans le cas précédent le domaine est

$$\Omega \subset \mathbb{R}^2, \quad \Omega = \{(x, y) : x^2 + y^2 < 1\}.$$

La représentation du domaine sur ordinateur se fait par découpage en polygones convexes, encore appelés "éléments":

$$\Omega \approx \Omega_h = \cup_{i \in I} T_i$$

Une triangulation est un tel recouvrement avec les propriétés suivantes (cf figure 1):

- La réunion des éléments recouvre le domaine.
- L'intersection de 2 éléments ne peut être que: vide, un sommet, une arête entière.
- Les coins du domaine doivent être des sommets de la triangulation.
- Les coins du domaine approché doivent être sur la frontière du domaine initial.

Remarquez qu'avec cette définition, $\Omega = \Omega_h$ si et seulement si le premier est polygonal.

2.3 Un exemple pour freefem

Nous allons définir la fonction f et construire une triangulation du cercle unité avec en écrivant dans un fichier de nom (par exemple) "trace.pde" le texte suivant:

```
/* fichier trace.pde */
n:=50;
border(1,0,2*pi,n) begin
  x := cos(t);
  y := sin(t);
end;
buildmesh(1000);
savemesh('trace.msh');

f = x * y;
plot(f);
save('trace.dta',f);
```

Ce texte est écrit dans la syntaxe Gfem. Il définit une frontière qui porte le numéro 1, dont le paramètre t dans la description paramétrique varie de 0 à 2π et qui sera découpé en segments par n sommets. Le mailleur automatique s'arrêtera si le nombre de sommets générés par l'algorithme Delaunay-Voronoi dépasse 1000 (cette incongruité facilite l'optimisation de la mémoire allouée à l'intérieur de freefem). Enfin le résultat sera stocké dans le fichier "trace.msh" selon un format qui sera détaillé dans le paragraphe suivant.

Le fichier des valeurs de f aux sommets de la triangulation s'appelle "trace.dta". Le format est

- Le nombre de sommets
- les valeurs de f sur chaque sommets sur une ligne par sommet

REMARQUE 4

Certaine version de freefem ayant prévue le cas f a valeur complexe, les valeurs de f aux sommets sont 2 nombres, la partie reel et la partie imaginaire. Verifiez quel version vous avez.

EXERCICE 3 Exécuter le programme Gfem précédent pour différentes valeurs de n .

Figure 1: Résultat

2.4 Fichier triangulation

Connaitre une triangulation c'est connaitre:

- chacun des sommets: par leurs coordonnées.
- Chacun les triangles: par les numéros des 3 sommets (en supposant qu'ils soient numérotés.

Mais il est commode de numéroter aussi les triangles de manière à pouvoir les identifier par leurs numéros seulement. Il est aussi commode d'associer un numéro logique à chaque sommet et à chaque triangle de manière à identifier des sous-ensembles de domaine comme l'ensemble des sommets ou des triangles ayant le même numéro logique.

```

5 4
0.0 0.0 1
2.0 0.0 1
2.0 1.0 1
0.0 1.0 1
1.0 0.5 0
1 5 4 0
1 2 5 0
2 3 5 0
3 4 5 0

```

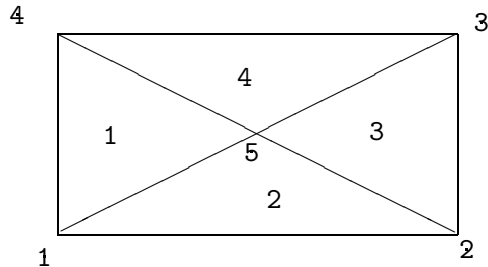


Figure 2: Une toute petite triangulation et le fichier associé a gauche. Dans le fichier on a d'abord le nombre de sommet puis le nombre de triangles puis sur chaque ligne les coordonnées de sommets ainsi qu'un entier d'appartenance a un frontiere; enfin sur les dernières lignes, les numéros des sommets de chaque triangles et le numéro d'appartenance une region pour chaque triangle.

Ainsi, nous ferons la convention que les sommets frontières ont un numéro logique positif.

On se propose de lire un fichier triangulation. Le format étant le suivant:

- Nombres de sommets. Nombres de triangles.
- Une ligne par sommet avec ses 2 coordonnées x, y et son numéro logique.
- Une ligne par triangle avec les numéros des 3 sommets et son numéro logique.

Les sommets et les triangles reçoivent le numéro correspondant à l'ordre d'entrée. (figure 2)

2.5 Relecture du fichier triangulation

En C l'équivalent du *record* Pascal s'appelle *struct*. En C++ c'est *classe*. Par exemple une triangulation étant faite de sommets (vertex) et de triangles, on définira la structure suivante:

```

// fichier lirmicro.cpp

#include <iostream.h>
#include <fstream.h>

```

```

//DECLARATIONS
class Vertex { public: float x,y; int where;};
class Triangle { public: int v[3]; int where;};

class Triangulation { public:
    int nv,nt;
    Vertex* v;
    Triangle* t;
    Triangulation(const char* path);
};

//IMPLEMENTATION
Triangulation::Triangulation (const char* path)
{
    int i,j,k,i1;
    ifstream file(path);

    file >> nv >>nt;
    v = new Vertex[nv];
    t = new Triangle[nt];
    for( i=0; i<nv; i++) file >> v[i].x >> v[i].y >> v[i].where;
    for( j=0; j<nt; j++)
        { for( k=0; k<3;k++){ file >> i1; t[j].v[k] = --i1;}
          file >> t[j].where;
        }
    file.close();
}

void main()
{
    Triangulation g("micro.msh");
}

```

REMARQUE 5

On a soustrait 1 aux numéros des sommets lus pour $t[j].v[k]$ en écrivant "- i1". Ceci est dû au fait que les tableaux en C/C++ commencent en 0 alors qu'ils commencent en 1 en FORTRAN. Nous adoptons la convention FORTRAN pour les fichiers car les programmeurs C connaissent cette difficulté et pas toujours les programmeurs FORTRAN.

2.6 Tracé avec gnuplot

Nous ne décrivons pas les multiples possibilités du logiciel du domaine public, gnuplot , mais seulement celles qu consistent à afficher les ordres de tracé contenu dans un fichier.

Par exemple, si le fichier 'ex.gnu' contient

```
0.0 0.0
0.0 0.1
0.1 0.05

0.1 0.05
0.0 0.0
```

alors les commandes

```
gnuplot
plot [-0.2,0,0.2] [-0.2,0.2] "ex.gnu" w l
quit
```

auront pour effet de tracer à l'écran, dans une fenêtre graphique, un triangle rectangle de base 0.1 et de hauteur 0.05. Le tracé se fait en commençant par le polygone ouvert des 3 premier point puis en trçant le dernier segment dont les extrémitées sont les 2 dernieres lignes. Autrement dit une ligne blanche provoque un "moveto" au point de la suivante alors que sinon c'est un "lineto" sur la suivante qui est effectué. La fenêtre permet de tracer tout objet s'inscrivant dans un carré [-0.2,0.2]x[-0.2,0.2]; en changeant ces dimensions on peut faire des "zoom".

Visiblement les fichiers de triangulation et les fichiers gnuplot sont différents. Sachant que gnuplot affiche tout couple de 2 lignes (séparées par une ligne vide) contenant les coordonnées de 2 points comme un segment de droite, la génération d'un fichier gnuplot pour tracer une triangulation à partir d'un fichier freefem est aisée:

```
void Triangulation::gnu(const char* filename) const
{
    ofstream f(filename);
    for(int i=0; i<nt; i++)
    {
        f << v[t[i].v[0]].x << "\t" << v[t[i].v[0]].y << endl
```

```

        << v[t[i].v[1]].x << "\t" << v[t[i].v[1]].y << endl
        << v[t[i].v[2]].x << "\t" << v[t[i].v[2]].y << endl
        << v[t[i].v[0]].x << "\t" << v[t[i].v[0]].y << endl
        <<endl;
    }
}

```

REMARQUE 6

Une fonction est du type "const" (mis en fin de declaration) lorsqu'elle n'affecte pas les champs de la classe.

Naturellement on devra rajouter la déclaration de la fonction "gnu" dans la classe Triangulation. La fonction main pour cette tache est donc

```

void main()
{
    Triangulation g("micro.msh");
    g.gnu("micro.gnu");
}

```

REMARQUE 7

Il existe un autre logiciel du domaine public plus élaboré que gnuplot: plotmtv.

EXERCICE 4

Se faire un utilitaire qui lit un fichier triangulation et un fichier de fonction au format freefem et génère un fichier de visualisation de la fonction au format gnuplot .

EXERCICE 5

- 1. Faire une triangulation avec freefem d'un carré unité avec un trou elliptique (les trous sont obtenus en changeant le sens de parcours de la frontière).
- 2. Changer le sens de parcours de l'ellipse, lui donner un numéro 0 et l'utiliser pour contrôler le raffinement du maillage dans cette région.
- 3. Raffiner le maillage du carré dans sa partie basse en introduisant une frontière interne de numéro 0 horizontale de longueur 1 et à une hauteur de 0.05. (Il faut couper les cotés verticaux en 2 segments pour assurer la jonction exacte avec la frontière interne).

2.7 Visualisation d'une fonction

Si maintenant le fichier gnuplot contient des lignes avec 3 valeurs pour (x,y,z) on peut l'afficher comme une surface en perspective. Pour interrompre le tracé d'une ligne polygonale il semble que sur certaine machine il faille 2 lignes blanches et non une seule.

Voici un exemple:

```
void main()
{
  Triangulation g("micro.msh");
  ofstream f("fmicro.gnu");
  float *u = new float[g.nv];

  for(int i=0; i<g.nv; i++)
    u[i] = g.v[i].x*g.v[i].x + g.v[i].y*g.v[i].y;

  for(int i=0; i<nt; i++)
  {
    f << g.v[t[i].v[0]].x << "\t" << g.v[t[i].v[0]].y
      << u[t[i].v[0]] << endl
      << g.v[t[i].v[1]].x << "\t" << g.v[t[i].v[1]].y
      << u[t[i].v[1]] << endl
      << g.v[t[i].v[2]].x << "\t" << g.v[t[i].v[2]].y
      << u[t[i].v[2]] << endl
      << g.v[t[i].v[0]].x << "\t" << g.v[t[i].v[0]].y
      << u[t[i].v[0]] << endl
      << endl << endl;
  }
}
```

Il faudra lancer gnuplot de cette façon

```
gnuplot
set parametric
splot "fmicro.gnu" w l
```

2.8 Complément: le debugger gdb

La mise au point des programmes est grandement facilitée par l'utilisation d'un debugger. Le plus simple sur gnu: gdb. Le plus sophistiqué: ddd. Les 2 étant compatibles nous donnons quelques indication sur gdb.

- Compiler le programme avec l'option "-g". puis executer gdb en donnant le nom du code généré. Exemple

```
g++ -g lirmicro.cpp
gdb a.out
```

- La liste des commandes est accessible en tapant "help". On peut verifier le code source en tapant "list" ou "l".
- executer le programme en mettant d'abord un breakpoint sur l'instruction 49 (Triangulation g("micro.msh");). Puis faire du pas a pas en tapant "s".
- occasionellement demander la valeur d'une variable en tapant "p " puis le nom de la variable.

Ceci donne la sequence suivante:

```
g++ -g lirmicro.cpp
gdb a.out
l
b 49
run
s
s
s
p nv
...
quit
```

2.8.1 La commande where

Connaitre la ligne de l'arret du programme en cas de "bug" ne suffit pas en général car une fonction peut être appelée en plusieurs endroits. Pour connaitre la sequence d'appel des fonctions, et donc quel appel de la fonction coupable au moment d'un "exit", le plus simple est d'utiliser la fonction "atexit" de la library stdlib:

```

#include <stdlib.h>
...
void myexit()
{
    cout << "Fin" << endl;
}
...
void main()
{
    atexit(myexit);
...
}

```

On mettra ensuite un "breakpoint" sur la ligne "cout<<"Fin"..." de sorte que le debugger s'arrêtera automatiquement sur cette ligne en cas de problème. En effet, l'effet de atexit(func) est de forcer l'exécution de la fonction func en cas d'interruption par le système genre écriture en zone protégée... Ensuite l'appel de "where" donne la liste d'emboîtement des fonctions et donc la position exacte du problème.

2.8.2 Passage de paramètre sous unix

Nous avons maintenant un utilitaire que nous pourrions appeler fem2gnu qui permet de lire un fichier .msh et de générer un fichier .gnu. Il serait pratique de pouvoir taper la commande

```
fem2gnu micro
```

pour pouvoir traduire le fichier "micro.msh" en un fichier "micro.gnu". Pour cela il suffit d'utiliser la construction suivante

```

main (int argc, char **argv)
{
    char* filein = new char[256];
    char* fileout = new char[256];
    if (argc != 2) //nb of strings on unix command
    {
        cout << "Check syntax of call to fem2gnu"<<endl;
        exit(0);
    }
    strcpy(filein,argv[1]); //second string copied

```

```

strcat(filein, ".msh"); //(1st string is prog name)
strcpy(fileout, argv[1]);
strcat(fileout, ".gnu"); // suffix added
Triangulation g(filein);
g.gnu(fileout);
}

```

2.9 Tracé d'une fonction de 2 variables

2.9.1 Principe: Interpolation P1

Si le domaine de la fonction f est triangulé par des triangles, le plus raisonnable est d'approcher la fonction par son "interpolée P1", c'est à dire la seule fonction qui soit

- continue,
- affine sur chacun des triangles
- égale à f aux sommets de la triangulation.

Théorème 1 *L'interpolé P1 d'une fonction continue est défini de manière unique par les valeurs aux sommets de la triangulation de la fonction.*

Démonstration

Soit x un point à l'intérieur d'un triangle T donné par ses coordonnées barycentriques:

$$x = \sum_{i=1,2,3} \lambda_i q^i, \quad \sum_{i=1,2,3} \lambda_i = 1,$$

on définit la valeur de l'interpolé P1 en x par

$$f_h(x) = \sum_{i=1,2,3} \lambda_i f(q^i).$$

Si x est sur une arête on définit l'interpolé par

$$f_h(x) = \alpha f(q^i) + (1 - \alpha) f(q^j), \quad \text{où} \quad x = \alpha q^i + (1 - \alpha) q^j.$$

On voit facilement que la définition interne coincide avec la définition sur les arêtes car si x tend vers une arête une des coordonnées barycentriques tend vers 0. Les *lignes de niveau* de f sont

$$L_\mu = \{x : f(x) = \mu\}.$$

Si on remplace f par son interpolé, les lignes de niveau seront des lignes polygonales, car sur chaque triangle ce sont des segments de droite, puisque la fonction est affine.

Un point d'intersection q de la courbe de niveau avec une arête e du triangle sera trouvé s'il existe i, j et a , avec

$$e = [q^i, q^j], \quad q = aq^i + (1-a)q^j, \quad f_h(q) = af_h(q^i) + (1-a)f_h(q^j) = \mu, \quad 0 \leq a \leq 1.$$

Le principe de tracé d'une ligne de niveau est donc le suivant:

```
for(int k = 0; k<nt; k++)
{ m=0;
  for(int i1=0; i1<3; i1++)
  {
    int j1 = (i1+1) % 3;
    int i = t[k].v[i1]->no, j = t[k].v[j1]->no;
    float a = (mu - f[j]) / (f[i] - f[j]);
    if((a>=0)&&(a<=1))
    { m++; if(m==1) move2(a * q[i].x + (1-a) * q[j].x,
                        a * q[i].y + (1-a) * q[j].y);
      else line2(a * q[i].x + (1-a) * q[j].x,
                a * q[i].y + (1-a) * q[j].y);
    }
  }
}
```

A ce programme il convient d'ajouter la vérification que les dénominateurs sont non nuls, que la fonction n'est pas identiquement constante, et de rajouter le tracé des frontières du domaine.

EXERCICE 6 (*Obligatoire*)

Ecrire un programme qui lit un fichier freefem de triangulation et un fichier de valeurs d'une fonction puis écrit un fichier gnuplot des tracés de ligne de niveau. On testera la présence de 2 paramètres sur la ligne de commande UNIX. S'il n'y en a qu'un on tracera la triangulation au lieu de la fonction

2.9.2 Estimations d'erreur

Dans [3] (Theoreme 6.1 p71) il est montré que l'erreur H1 entre l'interpolé et la fonction est en $O(h)$ alors que l'erreur L2 est en h au carré.

EXERCICE 7

Prendre 3 triangulations ayant chacune 4 fois plus de points que la précédente, écrire une fonction qui calcule l'erreur H1 et L2 dans le cas particulier ou $f=xy$. faire avec gnuplot un graphe du log de l'erreur en fonction du log de h . Sauver la figure au format Postscript et envoyer le resultat compresser par email.

3 LEÇON Les classes Triangulation

3.1 Première amélioration

Une amélioration est donnée par le programme suivant. D'abord la structure est enrichie; le tableaux des arêtes est défini, les sommets connaissent les numéros de tous les triangles auquel ils appartiennent (supp), ainsi que les numero des sommets auxquels ils sont reliés par une arête (mate). De même les triangles connaissent non seulement leurs 3 sommets mais aussi leurs 3 arêtes et les 3 triangles voisins par les arêtes.

```
class Vertex {
public:
    float x, y;           // cordinates
    int no, where;       // on which boundary
    int nsupp;           // nb of triangles which contains this vertex
    Triangle* supp;     //all triangles which contain the vertex
    int nmate;           // number of neighbor vertices
    Vertex* mate;       // all neighbors
};

class Triangle {
public:
    Vertex* v[3];       // the 3 vertices of the triangle
    Edge* e[3];         // pointer to the edges opposite each vertex
    int no, where;     // in which region
    float area;
};
```

```

};

class Edge { public:
    Vertex *in, *out;    // oriented by first to last vertex
    Triangle *left, *right; // triangles on each side of edge
    int where;          // on which curve (or boundary)
    float length;
};

class Grid {
public:
    int nt, nv, ne;      // nb of triangles, vertices and edges
    int nbholes;        // nb of holes in the domain
    int bdth;           // bandwidth
    Vertex* v;          // all vertices
    Triangle* t;        // all triangles
    Edge* e;            // all edges
    Grid(const char *path );
};

```

En quoi est-ce une amélioration? Les données sont mieux *encapsulées* dans leurs classes. Les triangles contiennent des pointeurs sur les vertex et non plus leurs numéros. Ainsi si la numérotation change on n'a pas besoin de mettre à jour les données. Le prix à payer est que les numéros des sommets et des triangles doivent être stockés afin de pouvoir y accéder (c'est le champs no), mais nous verrons qu'on peut en fait se débarrasser de ce champs et retrouver leurs numéros par leurs adresses.

3.2 Protection contre les dépassements de tableau

Pour se protéger contre les dépassements de tableau on utilise la classe *template* suivante

```

template <class T> class A{
public:
    T *cc;
    long size;

void init(long ssize);

```

```

T& operator [] (long i) const
{
    assert ( cc&&(i >= 0) && (i < size) );
    return cc[i];
}
A(long csize = 0)
{
    size = csize;
    if (size > 0 ) cc = new T[size];
    else cc = 0;
}
A(const A& a);
A& operator=(const A& a);
~A() { delete [] cc;size = 0; }
void destroy() { delete [] cc;size = 0; cc=0; }
int no( T* t) const { return t - cc;} // return place in array
};

//-----
template <class T> void A<T>::init(long ssize)
{
    assert( !cc && ssize );
    size=ssize;
    cc= new T[size];
    assert(cc != 0);
}
//-----
template <class T>  A<T>::A(const A<T>& a)
{
    if( a.cc && a.size )
    { size = a.size; cc = new T[size];
      for(int i=0; i<size;i++) cc[i] = a.cc[i];
    }
}
//-----
template <class T> A<T>& A<T>::operator=(const A<T>& a)
{
    assert( cc && a.cc && (a.size==size) );
    for(int i=0; i<size;i++) cc[i] = a.cc[i];
    return *this;
}

```

```
}
```

Toute création de tableau doit passer maintenant par les classes $A < type >$.

3.3 Deuxième amélioration

La classe triangulation devient

```
const nbholesmax=100;
class Vertex;
class Triangle;
class Edge;

/*****/
class Vertex {
public:
    float x, y;          // cordinates
    int where;          // on which boundary
    int nsupp;          // nb of triangles which contains this vertex
    A<Triangle*> supp; // all triangles which contain the vertex
    int nmate;          // number of neighbor vertices
    A<Vertex*> mate;     // all neighbors
};

/*****/
class Triangle {
public:
    Vertex* v[3]; // the 3 vertices of the triangle
    Edge* e[3];   // pointer to the edges opposite each vertex
    int where;    // in which region
    float area;
};

/*****/
class Edge { public:
    Vertex *in, *out; // oriented by first to last vertex
    Triangle *left, *right; // triangles on each side of edge
    int where; // on which curve (or boundary)
```

```

    float length;
};

/*****
class Grid {
public:
    int nt, nv, ne; // nb of triangles, vertices and edges
    int nbholes;   // nb of holes in the domain
    int bdth;      // bandwidth
    A<Vertex> v;    // all vertices
    A<Triangle> t;  // all triangles
    A<Edge> e;      // all edges
Grid(){nt=nv=ne=0;nbholes=nbholesmax;}// default constructor
Grid(const char *path );    // reads a triangulation
void save(const char* path) const; //save mesh in Gfem format
int no(Triangle* tt) const { return t.no(tt);}
int no(Vertex* tt) const { return v.no(tt);}
int no(Edge* tt) const { return e.no(tt);}//place in e of tt
};

Grid::Grid(const char *path ):v(),t(),e()
{// reads a triangulation in MacGfem format
    int i,j, ii;
    float x,y;
    ifstream file(path);

    assert(!file.fail());
    file >> nv >> nt;
    nbholes =nbholesmax; // not in macgfem format
    v.init(nv);
    t.init(nt);
    e.init(nv+nt+nbholes-1);
    for( i=0; i<nv; i++ )
        file >> v[i].x >> v[i].y >> v[i].where;
    for( i=0; i<nt; i++ )
    {
        for(j=0;j<3;j++){ file >> ii; t[i].v[j] = &v[ii-1];}
        file >> t[i].where;
        t[i].area = ((t[i].v[1]->x - t[i].v[0]->x)
                    * (t[i].v[2]->y - t[i].v[0]->y)

```

```

        - (t[i].v[2]->x - t[i].v[0]->x)
        * (t[i].v[1]->y - t[i].v[0]->y))/2;
    }
    file.close();
}

//-----
void Grid::save(const char *path ) const
{
    // write a triangulation in MacGfem format
    int i,j=0;
    ofstream file(path);

    file << nv <<" " << nt<<endl;
    for( i=0; i<nv; i++ )
        file << v[i].x <<" " << v[i].y <<" " << v[i].where<<endl;
    for( i=0; i<nt; i++ )
        file << no(t[i].v[0])+1 <<" " << no(t[i].v[1])+1
        <<" " << no(t[i].v[2])+1<<" " << j<<endl;
    file.close();
}

```

3.4 Construction du tableau des arêtes

En faisant une boucle sur les triangles et une boucle sur les 3 cotés de chaque triangle on décrit certainement toutes les arêtes. Le principe de la construction est donc

```

for(int k=0;k<g.nt;k++)
    for(int iloc=0;iloc<3;iloc++)
    {
        e[1].in = t[k].v[iloc];
        e[1++].out = t[k].v[(iloc+1)%3];
    }

```

Toutefois ce procédé décrivant 2 fois les arêtes internes, celles-ci se retrouvent 2 fois dans le tableau e.

Il faut donc prendre la convention qu'on ne mettra l'arête dans e seulement si, par convention, le numéro du sommet "in" est plus petit que le numéro du sommet "out":

```

for(int k=0;k<g.nt;k++)

```

```

for(int iloc=0;iloc<3;iloc++)
{
  Vertex& vin = t[k].v[iloc], vout = t[k].v[(iloc+1)%3];
  if(g.no(vin) < g.no(vout))
  {
    e[l].in = t[k].v[iloc];
    e[l++].out = t[k].v[(iloc+1)%3];
  }
}

```

(Se reporter au troisième exercice ci-dessous pour le mécanisme de la fonction `no()` qui rend le numéro d'un sommet ou d'un triangle.)

Mais alors certaines arêtes frontières, qui elles ne sont décrites qu'une fois, risquent de ne pas rentrer dans le tableau `e`. Il ne faut donc faire ce test que si l'arête n'est pas frontière.

Par convention, une arête frontière est une arête dont les 2 sommets sont frontières. Donc

```

for(int k=0;k<g.nt;k++)
  for(int iloc=0;iloc<3;iloc++)
  {
    Vertex& vin = t[k].v[iloc], vout = t[k].v[(iloc+1)%3];
    if((vin.where && vout.where) || (g.no(vin) < g.no(vout)))
    {
      e[l].in = t[k].v[iloc];
      e[l++].out = t[k].v[(iloc+1)%3];
    }
  }
}

```

REMARQUE 8

Malheureusement il existe des arêtes qui vérifient la définition "frontière" et qui pourtant sont internes dans le sens qu'elles appartiennent à deux triangles distincts. On ne peut, hélas, pas résoudre ce cas pathologique et c'est le format `freefem` qui est en défaut! Il faut soit prévenir l'utilisateur, soit faire une vérification a posteriori du tableau `e` pour retirer ces arêtes qui sont en double exemplaire.

3.5 Les Exercices de la leçon

EXERCICE 8 (*Obligatoire*)

Ecrire une fonction de la classe Grid qui numérote les arêtes. Renvoyer par email la fonction. Appelez la en fin de lecture de la triangulation dans le constructeur Grid.

EXERCICE 9

Recopier ces programmes dans des fichiers: Adapter le programme de génération d'un fichier gnu pour visualiser la triangulation à la nouvelle structure de donnée.

- La classe template A dans un fichier: aarray.h.
- Les classes de la triangulation dans un fichier: grid.h.
- Les implémentations des fonctions dans un fichier: grid.cpp.
- Le programme principal dans un fichier: lecon2.cpp
- Les données d'une triangulation de votre choix générée par freefem dans un fichier: lecon3.msh. Compiler et exécuter. Envoyer par email les fichier sources en faisant une archive 'tar' compressée, puis uuencode (ce dernier transcode toutes les bytes en caractère texte, permettant d'envoyer n'importe quoi comme du texte). Pour ce faire le plus simple est de mettre les sources dans un dossier (directory) et d'exécuter les commandes UNIX:

```
tar -cvf lecon3.tar *
compress lecon3.tar
```

REMARQUE 9

Linux en général permet de faire les 2 opérations en une seule commande: `"tar czvf lecon2.tar.Z *"`

Si votre email ne permet pas d'envoyer un fichier contenant des caractères de contrôle, il faut en plus faire uuencode sur le fichier (voir le manuel accessible par la commande `"man uuencode"`).

REMARQUE 10

Pour pouvoir executer un programme en plusieurs fichiers il faut faire une edition de lien ("link"): faire un makefile, c'est a dire un fichier contenant les ordres nécessaires puis taper la commande UNIX: make. Ici le makefile sera un fichier text appelé "makefile". Se reporter à l'exemple donné à la fin de la leçon 1.

EXERCICE 10

Ecrire la modification qui retire les arêtes frontières apparaissant éventuellement 2 fois dans le tableau e. Tester sur un carré avec peu de point (normalement freefem genere 2 de ces arêtes pathologiques).

EXERCICE 11

Existe t il une solution au problème "nbholes=100". C'est à dire que connaissant le nombre de frontières fermées, peut on en déduire le nombre de trous et si oui construire le programme. La construction des composantes simplement connexes de la frontière (frontières fermées) se fait par une boucle intelligente sur les arêtes, (éventuellement avec tri?)

EXERCICE 12

Si on rajoute à la dernière ligne de la déclaration de la classe template classe A la ligne de définition d'une fonction no()

```
int no( T* t){ return t - cc;}
```

et si dans la classe Grid on rajoute les fonctions

```
int no( Triangle* tt){ return t.no(tt);}  
int no( Vertex* vv){ return v.no(vv);}
```

alors on a plus besoin des champs .no pour les triangles, sommets... On peut remplacer quand nécessaire x.no par x.no(). Expliquer comment et pourquoi ca marche. On pourra consulter le fichier "grid.cpp" du pro-

gramme "tnfem" (voir le site ftp (accès aussi par sa page personnelle dans www.ann.jussieu.fr) pour voir une implémentation complète des fonctions de numérotation d'arête et de voisinage des triangles et des sommets.

4 LEÇON Un programme d'Elements Finis P1

4.1 Introduction

Soit à résoudre

$$-\Delta\varphi = f \text{ dans } \Omega, \quad \varphi(\vec{x}) = g(\vec{x}), \quad \forall \vec{x} \in \Gamma = \partial\Omega.$$

Nous allons minimiser l'énergie du système:

$$\min_{\varphi \in H} E(\varphi) = \frac{1}{2} \int_{\Omega} |\nabla\varphi(\vec{x})|^2 d\vec{x} - \int_{\Omega} f(\vec{x})\varphi(\vec{x})d\vec{x},$$

sur le sous espace $H \subset H^1(\Omega)$, formé des fonctions de carré intégrable, dont les dérivées sont de carré intégrable et qui valent g sur la frontière du domaine.

La discrétisation est obtenue en remplaçant H par un espace de dimension finie:

$$H_h = \{w_h \in C^0(\Omega_h) : w_h|_{T_k} \in P^1, \quad w_h|_{\Gamma} = g_h\}$$

c'est à dire l'espace des fonctions continues, affines sur chaque triangle d'une triangulation et égales à l'interpolé P1 de g sur le bord.

Une base de cet espace est obtenue en considérant l'ensemble des fonctions chapeaux qui valent 1 ou 0 sur les sommets de la triangulation et qui ne sont pas associée à un sommet sur la frontière.

$$w_h \in C^0(\Omega_h), \quad w_h|_{T_k} \in P^1, \quad w^j(q^i) = 1 \text{ si } i = j \text{ et } 0 \text{ sinon.}$$

On designe par I l'ensemble des indices des sommets internes et par J l'ensemble des indices des sommets frontières: Donc le problème discret est le suivant:

$$\min_{\varphi_i: i \in I} E(\varphi_h) : \varphi_h(\vec{x}) = \sum_{i \in I} \varphi_i w^i(\vec{x}) + \sum_{i \in J} g_i w^i(\vec{x}).$$

4.2 Algorithme du gradient conjugué

Comme il s'agit d'un problème de minimisation sans contrainte, on peut utiliser l'algorithme suivant dans lequel N_h désigne le nombre de sommet de la triangulation:

- 0 choisir la précision epsilon, le nombre maximal d'iterations mMax, choisir une initialisation $\Phi^0 = (\varphi_1^0, \dots, \varphi_{N_h}^0)$ de Φ vérifiant $\varphi_i = g(q^i)$ pour tout sommet du bord, poser $m=0$, $H^{-1} = (H_1^{-1}, \dots, H_{N_h}^{-1})$, avec $H_k^{-1} = 0, \forall k \in \{1, \dots, N_h\}$ et définir $l^{-1} = 1$;

- 1 calculer, pour tous les indices $k \in \{1, \dots, N_h\}$,

$$G_k^m = \frac{\partial E}{\partial \varphi_k}(\Phi^m), \quad \Phi^m = (\varphi_1^m, \dots, \varphi_{N_h}^m);$$

poser $G^m = (G_1^m, \dots, G_{N_h}^m)$ et définir

$$l^m = \|G^m\|_{R^{N_h}}^2 = \sum_{k=1}^{N_h} (G_k^m)^2;$$

- 2 calculer $H^m = -G^m + \gamma H^{m-1}$, où $H^m = (H_1^m, \dots, H_{N_h}^m)$, avec $\gamma = \frac{l^m}{l^{m-1}}$;
- 3 poser $\Phi(\rho) = \Phi^m + \rho H^m$, puis calculer le réel ρ^m qui minimise sur \mathbb{R} la fonction $E(\Phi(\rho))$;
- 4 Poser $\Phi^{m+1} = \Phi^m(\rho^m)$, incrémenter m de 1 et retourner à l'étape 1 jusqu'à ce que $l^m < \epsilon$ ou $m > mMax$ (dans ce cas, écrire que le programme ne marche pas).

La valeur minimale ρ^m recherchée se calcule explicitement par la résolution d'une équation du deuxième degré car E est quadratique et nous obtenons

$$\rho^m = -\frac{(G^m, H^m)_{R^{N_h}}}{\|\nabla H^m\|_{L^2(\Omega_h)}^2},$$

où le symbole $(\cdot, \cdot)_{R^{N_h}}$ désigne le produit scalaire associé à la norme euclidienne.

On rappelle que

$$\frac{\partial E_h}{\partial \varphi_k}(\Phi) = \int_{\Omega_h} \nabla \varphi_h(\vec{x}) \cdot \nabla w^k(\vec{x}) d\vec{x} - \int_{\Omega_h} (f w^k)(\vec{x}) d\vec{x}.$$

4.3 Structures de données

4.3.1 La Classe Vector

Un vecteur de taille "size" est un tableau (donc une sous classe de A<float>) de valeurs sur lesquelles on peut faire les opérations des espaces vectoriels: additions soustraction multiplication par un scalaire, produit scalaire, norme... L'algèbre des scalaires est pour nous les float.

```

class Vector: public A<float>{
public:
    Vector(int ssize=0): A<float>(ssize){}
    Vector(Vector& v): A<float>(v){}
    Vector& operator = (const Vector& v1);
    Vector& operator += (const Vector& v1);
    Vector operator + (const Vector& v1)
        { Vector v2(*this); v2 += v1; return v2;}
    Vector& operator -= (const Vector& v1);
    Vector operator - (const Vector& v1) const;
    Vector operator* (const float a) const;
    friend Vector operator*(const float a, const Vector& v1);
    Vector operator/ (const float a) const;
    float scal(const Vector& v1) const;
};

//-----
Vector& Vector::operator= (const Vector& v1)
{
    assert(size==v1.size);
    for(int i=0; i<size;i++) cc[i] = v1.cc[i];
    return *this;
}
//-----
Vector& Vector::operator+= (const Vector& v1)
{
    for(int i=0; i<size;i++) cc[i] += v1.cc[i];
    return *this;
}
//-----
Vector Vector::operator* (const float a) const
{
    Vector v2(size);
    for(int i=0; i<size;i++) v2.cc[i] = a * cc[i];
    return v2;
}
//-----
Vector operator* (const float a, const Vector& v1)
{
    Vector v2(v1.size);

```

```

    for(int i=0; i<v1.size;i++) v2.cc[i] = a * v1.cc[i];
    return v2;
}
//-----
float Vector::scal (const Vector& v1) const
{
    float s=0;
    for(int i=0; i<size;i++) s += v1.cc[i] * cc[i];
    return s;
}

```

Notez les 2 fonctions operator* correspondant à a*v et v*a lorsque a est un float et v est un vecteur.

4.3.2 Fonctions P1

Une fonction P1 est définie par ses valeurs aux sommets de la triangulation. Le coeur de la classe P1 sera donc une sous classe de la classe Vector

Les fonctions P1 sont susceptibles d'autres opérations:

- Lecture et écriture au format freefem,
- integration, dérivation,
- tracé graphique...

Mais compte tenu de la suite nous n'introduisons que les fonctions de lecture d'écriture sur fichier, et d'integration pour calculer:

$$\int_{\Omega} |\nabla f|^2 d\vec{x}, \quad \int_{\Omega} f(\vec{x})g(\vec{x})d\vec{x}$$

```

float deter(const float a,const float b,const float c,
            const float d,const float e,const float f)
{ float g = (a-c)*(e-f) - (b-c)*(d-f);
  return g;
}
/*****/
class P1: public Vector
{
public:
    Grid* g;
}

```

```

P1(Grid* gg): g(gg), Vector(gg->nv){}
void load(const char* filename);
void save(const char* filename) const;
float intgrad2() const; //int gradf.gradf dx
};

//-----
void P1::save(const char* path) const
{ // writes a P1 continuous function in freefem format
  ofstream file(path);
  int nv = size;
  assert(!file.fail());
  file << nv << endl;
  for(int i=0; i<nv; i++) file << cc[i] << endl;
  file.close();
}

//-----
void P1::load(const char* path)
{ // read a P1 continuous function in freefem format
  ifstream file(path);
  int nv ;
  assert(!file.fail());
  file >> nv;
  assert(nv==size);
  for(int i=0; i<nv; i++) file >> cc[i];
  file.close();
}

//-----
float P1::intgrad2() const
{
  float x[3], y[3], phi[3], gradFx, gradFy, ee;
  int k, iloc, j;

  ee = 0.;
  for(k=0; k<g->nt; k++)
  { float a1 = 0.5/g->t[k].area;
    for(int jloc=0; jloc<3; jloc++)
    { j = g->no(g->t[k].v[jloc]);

```

```

    x[jloc] = g->v[j].x; y[jloc]= g->v[j].y;
    phi[jloc] = cc[j];
}
gradFx = a1 * deter(phi[0],phi[1],phi[2],y[0],y[1],y[2]);
gradFy = a1 * deter(x[0],x[1],x[2],phi[0],phi[1],phi[2]);
ee += g->t[k].area * (gradFx*gradFx + gradFy*gradFy);
}
return ee;
}

```

4.3.3 La classe Problem

Un problème de résolution d'une EDP est défini par

- Un operateur, ici le Laplacien,
- une solution "sol" de type P1.
- un second membre: rhs,
- des conditions aux bords ici de Dirichlet, stockées dans "sol" en entrée,
- une méthode de résolution: ici le gradient conjugué
- les méthodes nécessaire au gradient conjugué: l'évaluation de la dérivée de lénergie par rapport aux degrés de liberté de la solution

```

class Laplace{ public:
    Grid g;
    P1 sol; // contains bdy cond on entry
    P1 rhs; // right hand side

    Laplace(const char* meshfile, const char* rhsfile,
            const char* dirichletfile):
        g(meshfile), rhs(&g), sol(&g)
        { rhs.load(rhsfile), sol.load(dirichletfile);}
    void solvegradconj(const int niter, const float precise);
    Vector derivener() const;
};

//-----
void Laplace::solvegradconj(const int niter, const float eps)

```

```

{
int i,m;
float g2hm=1, g2h, eps1, ro, gamma, E1;
P1 hconj(&g);
Vector hconjm(g.nv), gradE(g.nv);

for(m=0; m<niter; m++)
{
gradE = derivener();
for(i=0;i<g.nv;i++) if(g.v[i].where) gradE[i] = 0;
      g2h = gradE.scal(gradE);
if(m==0) { eps1 = eps*g2h; gamma = 0;}
      else gamma = g2h / g2hm;
      for(i=0;i<g.nv;i++)
          hconj[i] = -gradE[i] + gamma * hconjm[i];
      ro = - gradE.scal(hconj) / hconj.intgrad2();
      cout << m <<"\t"<<ro<<"\t"<<g2h<<endl;
      if(g2h<eps1) break;
      for(i=0;i<g.nv;i++)
          {
              sol[i] += ro* hconj[i];
              hconjm[i] = hconj[i];
          }
      g2hm = g2h;
  }
}

Vector Laplace::derivener() const
{
int i,j,k,jloc;
float x[3], y[3], phi[3];
float gradFx,gradFy,gradGx, gradGy;
Vector r(g.nv);
for(i=0;i<g.nv;i++) r[i]=0;
for (k=0; k<g.nt; k++)
{
    Triangle& tk = g.t[k];
    float a1 = 0.5/ tk.area;
    float f0 = rhs[g.no(tk.v[0])] + rhs[g.no(tk.v[1])]
              + rhs[g.no(tk.v[2])];

```

```

for (jloc=0; jloc<3; jloc++)
{
    j = g.no(tk.v[jloc]);
    r[j] -= ( rhs[j] + f0 ) * (tk.area/12);
    x[jloc] = g.v[j].x;  y[jloc]= g.v[j].y;
    phi[jloc] = sol[j];
}
gradFx = a1 * deter(phi[0],phi[1],phi[2],y[0],y[1],y[2]);
gradFy = a1 * deter(x[0],x[1],x[2],phi[0],phi[1],phi[2]);
for(i=0;i<3;i++)
{ int i0= i==0, i1= i==1, i2= i==2,
  gradGy = a1 * deter(x[0],x[1],x[2],i0,i1,i2);
  gradGx = a1 * deter(i0,i1,i2,y[0],y[1],y[2]);
  r[g.no(tk.v[i])] += tk.area * (gradFx*gradGx + gradFy*gradGy);
}
}
return r;
}

void myexit();

void myexit()
{
    cout << "fin" << endl;
}

void main()
{
    atexit(myexit);
    Laplace p("circle.msh", "one.dta", "zero.dta");
    p.solvegradconj(1000, 1e-4);
    p.sol.save("sol.dta");
}

```

4.3.4 Remarque sur l'assemblage

Toutes les integrales sont calculées par la technique de l'assemblage. Par exemple soit a calculer $\int(f \, wk)$. On fera une boucle sur k puis une boucle sur l de 0 a nt-1:

```
for(int k=0; k<nv; k++) I[k] = 0;
```

```

for(int k=0; k<nv; k++)
  for(l=0; l<nt; l++)
    I[k] += Ikl;

```

ou I_{kl} est l'intégrale sur T_l de fw_k .

C'est interdit car cela fait $O(N*N)$ opérations ($N=nt$ ou nv). En revanche si on change l'ordre des boucles:

```

for(int k=0; k<nv; k++) I[k] = 0;
for(l=0; l<nt; l++)
  for(int k=0; k<nv; k++)
    I[k] += Ikl;

```

alors on remarque que I_{kl} est nul si k n'est pas un numero de sommet de T_l et donc la 3eme boucle peut se limiter a $k=no(t[l].v[kloc])$ avec $kloc=0,1,2$, ce qui donne

```

for(int k=0; k<nv; k++) I[k] = 0;
for(l=0; l<nt; l++)
  for(int kloc=0; kloc<3; kloc++)
  {
    int k=no(t[l].v[kloc]);
    I[k] += Ikl;
  }

```

4.4 Les Exercices de la leçon

EXERCICE 13

Faire avec freefem les fichiers "circle.msh", "one.dta", "zero.dta" contenant la triangulation d'un cercle, la fonction valant 1 sur tous les sommets et la fonction valant zero sur tous les sommets. Exécuter le programme et visualiser la solution avec freefem. (Eventuellement rajouter le lien avec rgraph.h et faire une fonction `P1::show()` qui visualise le résultat "online").

EXERCICE 14

Comparer avec la solution analytique et faire une fonction qui calcule l'erreur L2 et l'erreur H1. Faire un tableau des erreurs obtenues pour 3 triangulations ayant chacune 4 fois plus de points que la précédente. Rajouter les conditions de Neuman aux bords (Voir remarque ci-dessous).

REMARQUE 11

Si on a des conditions de Neuman sur une partie du bord:

$$\frac{\partial \phi}{\partial n} = r \text{ sur } \Gamma_1 \subset \Gamma$$

il suffit de rajouter une integrale de bord dans la fonctionnelle à minimiser et une autre intégrale de bord dans le calcul de la dérivée de la fonctionnelle. C'est à dire remplacer

$$\int_{\Omega} f \phi d\vec{x} \text{ et } \int_{\Omega} f w^i d\vec{x} \text{ par } \int_{\Omega} f \phi d\vec{x} + \int_{\Gamma} r \phi d\gamma \text{ et } \int_{\Omega} f w^i d\vec{x} + \int_{\Gamma} r w^i d\gamma$$

ou r est prolongée par zero sur le reste du bord.

Mais le problème est alors de reconnaitre informatiquement le bord Dirichlet du bord Neuman. Une solution consiste a utiliser une fonction P1 sol0 contenant les conditions de Dirichlet avec la définition suivante:

Si sol0[i]=0 alors i n'est pas un point de Dirichlet. Si sol0[i]!=0 alors i est un point de Dirichlet.

Dans le cas d'une condition de Dirichlet nulle il faudra remplacer sol[i]=0 par sol[i]=eps en ce point, eps étant un tout petit nombre.

EXERCICE 15 (*Obligatoire*)

En plus des conditions de Neumann faire les modifications nécessaire pour pouvoir résoudre le problème

$$a\phi - \nabla \cdot \nu \nabla \phi = f \text{ dans } \Omega \phi = g \text{ sur } \Gamma - \Gamma_1, \quad \frac{\partial \phi}{\partial n} = r \text{ sur } \Omega_1$$

où a, ν sont des constantes positives.

Comme dans l'exercice ci-dessus il faudra modifier la fonctionnelle et sa dérivée en conséquence en rajoutant le xcoefficient nu et les intégrales

$$\int_{\Omega} \frac{a}{2} \phi^2 d\vec{x}, \quad \int_{\Omega} a \phi w^i d\vec{x}.$$

4.5 Directives pour cet exercice

Les modifications sont a faire dans

- P1::intgrad2
- Laplace::derivener
- Laplace::solvgradconj
- Laplace::Laplace

4.5.1 Dans P1::intgrad2

Cette fonction calcule l'integrale sur le domaine du carre du gradient de la fonction. Il faut maintenant que cela soit multiplié par nu et qu'il lui soit ajouté l'integrale sur le domaine de a fois la fonction au carre. Les constantes nu et a doivent donc etre des parametres de la fonction.

```
P1::intgrad2(const float& nu, const float& a)
```

Le principe de calcul de la 2eme integrale I est le suivant

```
I=0;
for(int k=0; k<g->nt; k++)
{
    int i0 = g->no(g->t[k].v[0]);
    int i1 = g->no(g->t[k].v[1]);
    int i2 = g->no(g->t[k].v[2]);
    I+= g->t[k].area * a * ( (cc[i0]+cc[i1])*(cc[i0]+cc[i1])
        + (cc[i1]+cc[i2])*(cc[i1]+cc[i2]) + (cc[i2]+cc[i0])*(cc[i2]+cc[i0])
        )/12;
}
```

4.5.2 Changement dans Laplace::derivener

On doit rajouter

$$I[k] = \int_{\Omega} (a w w^k) \text{ et } J[k] = \int_{\Gamma} (g w^k)$$

Le calcul de $I[k]$ se fait exactement comme le calcul de l'intégrale sur Ω de f_{wk} qui est déjà fait au début de la fonction. Il suffit de recopier et de remplacer `rhs[]` par `a * sol[]`

Le calcul de $J[k]$ est plus compliqué. Il faut impérativement garder en mémoire qu'on ne peut faire $O(N^2)$ opérations donc pas de double boucles imbriquées. Le principe de base est l'assemblage: plutôt que de faire une boucle sur k puis une boucle sur les arêtes frontières (faisable car il n'y a que 2 arêtes qui contribuent à $J[k]$ pour un cas donné) on échange l'ordre des boucles et on fait une boucle sur les arêtes puis une boucle sur les k qui donnent des contributions non nulles à J dues à l'arête. Pour que ça marche il faut initialiser à zéro le tableau J . Pour faire une boucle sur les arêtes, on fait

```
for(int l=0; l<g->nt; l++)
  for(int iloc=0; iloc<3; iloc++)
  { Vertex* vi = g->t[l].v[iloc];
    Vertex* vj = g->t[l].v[(iloc+1)%3];
    if(vi->where && vj->where)
    { // on est sur une arête du bord, calculer sa longueur
      // mettre ici la suite
    }
  }
}
```

Donc le calcul de la contribution sur l'arête à l'intégrale se fait par

```
float Ik = (neuman[g->no(vi)] + neumann[g->no(vj)]) * longueur/4;
```

car w_k vaut $1/2$ au milieu de l'arête. Enfin cette contribution ira dans les seuls indices k pour lesquels w_k est non nul au milieu de l'arête c'est à dire i et j .

```
I[g->no(vi)] += Ik;
I[g->no(vj)] += Ik;
```

4.5.3 Modification dans `Laplace::solvgradconj`

Une seule chose à changer: maintenir `hconj[k]` nul si k est un indice de vertex Dirichlet. Il faut donc remplacer le test

```
"if(v[i].where)" par "if(fabs(u0[i])> precise)"
```

ou $u0$ contient la condition de Dirichlet. On rappelle la convention $u0[i]=0$ entraine qi n'est pas un point de Dirichlet

On rappelle aussi que les constructeurs sont fait pour que $u0$ soit dans sol en entree dans solvgradconj. Il faut donc le stocker dans un tableau de travail (une fonction P1) avant de faire des iterations qui change sol.

4.5.4 Remarque sur I_{kl}

Dans l'expression précédente, si $i0,i1,i2$ sont les numéros des 3 sommets du triangle T_k

$$I_{kl} = (t[1].area/3) * (f[i0] + f(i1))/2 *(1/2) *(!(i2==k)) \\ + (f[i1] + f(i2))/2 *(1/2) *(!(i0==k)) \\ + (f[i2] + f(i0))/2 *(1/2) *(!(i1==k));$$

car w_k vaut 0 sur l'arête opposée a k et $1/2$ sur les 2 autres arêtes. Cette expression se simplifie en

$$I_{kl} = t[1].area * (f[i0] + f(i1) + f[i2])/12 \\ + t[1].area * f[k] / 12;$$

4.5.5 Changement dans Laplace::Laplace

Ce constructeur a maintenant la charge aussi d'initialiser a et nu ainsi que de lire le fichier contenant les conditions de Neumann. Il aura donc 3 parametres e plus. Ces dernières seront stockées dans une fonction P1, neum, par exemple, qui sera un champs supplémentaire e la classe Laplace. Et de même pour a et nu .

4.5.6 Gén/’eration de données

Voici un exemple de programme freefem générant les données nécessaires pour exécuter le programme. On remarquera l'usage de la fonction P1 "ib" qui vaut le numéro logique en chaque sommet: zero si interne, numéro de la frontière sinon.

```
/* fichier circle.pde */  
n:=50;
```

```

border(1,0,2*pi,n) begin
  x := cos(t);
  y := sin(t);
end;
border(2,0,2*pi,n/2) begin
  x := 0.3*cos(-t);
  y := 0.3*sin(-t);
end;
buildmesh(1000);
savemesh('circle.msh');

f = 1;
plot(f);
save('one.dta',f); // right hand side
f = 1.0e-20 * (ib==1);
plot(f);
save('zero.dta',f); // Dirichlet conditions on 1

f = -1*(ib==2);
plot(f);
save('neumann.dta',f); // Neumann condition

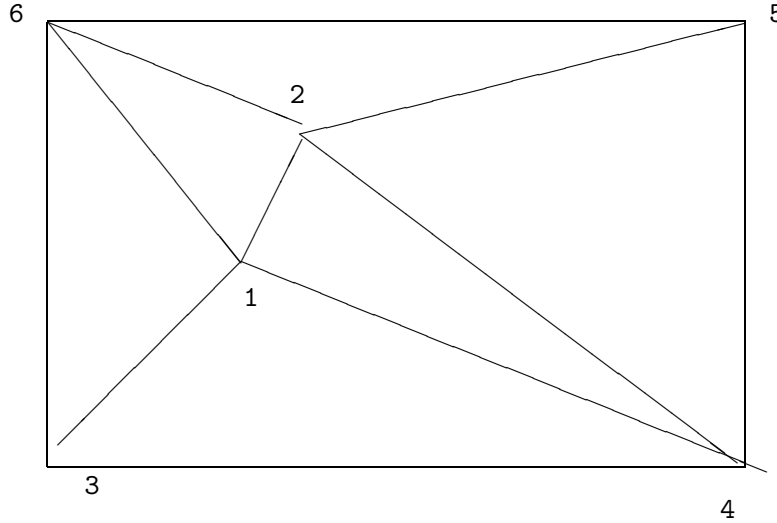
```

4.5.7 Vérification

La meilleure façon de s'assurer que le programme donne le bon résultat est de partir d'une fonction connue, par exemple $u = x^2 + y^2$ et de choisir les données f, u_0, g , pour retrouver cette solution. Ainsi avec $a=nu=1$, $f = x^2 + y^2 - 4$, $u_0 = u$ et sur un cercle intérieur de rayon r , $g = \text{grad } u \cdot n = (2x, 2y) \cdot (-x/r, -y/r) = -2$.

EXERCICE 16

Faire les simulations du chapitre 1 du livre Lucquin-Pironneau[?] avec ce module quand l'EDP est du bon type.



5 LEÇON Triangulation Automatique

5.1 Triangulation automatique d'un rectangle.

5.1.1 Le problème:

Soit un rectangle et N points donnés strictement internes au rectangle. Il sera commode de numéroter les points $(q_1, \dots, q_N, q_{N+1}, q_{N+2}, q_{N+3}, q_{N+4})$ où $(q_{N+1}, q_{N+2}, q_{N+3}, q_{N+4})$ sont les sommets du rectangle.

5.1.2 Algorithme

On maintient à jour une liste de triangle qui à tout instant forme une triangulation du rectangle. Celle-ci est formée au départ des 2 triangles obtenus en divisant le rectangle par une de ses diagonales.

On prend les points un par un dans l'ordre de leur numéro. Deux cas possibles:

Ou bien le nouveau point est interne à l'un des triangles T_k de la liste ou bien il est sur la frontière de 2 triangles T_i, T_j de la liste.

Dans le premier cas, retirer T_k de la liste et rajouter à la liste les 3 triangles obtenus en divisant T_k par les droites joignant le point au 3 sommets de T_k .

Dans le deuxième cas retirer T_i et T_j de la liste et rajouter les 4 triangles obtenus en divisant T_i par le nouveau point et le sommet opposé et de même

pour T_j .

5.1.3 Le critère de Delaunay

En fin d'algorithme on obtient une triangulation du rectangle qui passe par tous les points donnés. Mais la triangulation n'a pas une "bonne tête".

Une triangulation sera dite de Delaunay si tout cercle circonscrit à un triangle ne contient aucun sommet en son intérieur.

De cette caractérisation géométrique découle la propriété suivante, dont la preuve est immédiate: Soient q_1, q_2, q_3, q_4 , 4 points du plan non co-

cycliques et formant un quadrilatère convexe ; alors

(i) il existe 2 triangulations de l'enveloppe convexe de ces 4 points : la triangulation de Delaunay et une autre triangulation , qui se déduisent l'une de l'autre en changeant de diagonale dans le quadrilatère q_1, q_2, q_3, q_4 .

(ii) la triangulation de Delaunay est celle qui maximise la borne inférieure des angles des triangles.

5.1.4 Algorithme

Pour rendre une triangulation quelconque "Delaunay":

- Pour chaque arête non-frontière, trouver les 2 triangles adjacents T_k, T_l de sommets notés q_1, q_2, q_3, q_4 . Pour l'un des triangles, T_k par exemple, vérifier que le 4eme point n'appartient pas au cercle circonscrit au triangle. Si tel n'est pas le cas, modifier les 2 triangles T_k et T_l en changeant de diagonale dans le quadrilatère q_1, q_2, q_3, q_4 .
- Refaire cette boucle sur les arêtes jusqu'à ce qu'il n'y ait plus d'échange de diagonale.

Si on choisit de démarrer cet algorithme par l'arête opposée au plus grand angle de la triangulation, alors le procédé est convergent, puisque le plus grand angle décroît strictement à chaque itération.

5.2 Triangulation d'un domaine convexe

Soit un domaine convexe D donné par ses arêtes frontières et des points intérieurs. On désigne par q_1, q_2, \dots, q_N l'ensemble des points.

5.2.1 Algorithme

Choisir 4 points q_{N+1}, \dots, q_{N+4} formant un rectangle et très éloigné du barycentre des points $q_1 \dots q_N$.

Appliquer l'algorithme précédent.

Retirer tous les triangles qui ont des q_{N+1}, \dots, q_{N+4} comme sommet. Si

les q_{N+1}, \dots, q_{N+4} sont assez éloignés alors cet algorithme marche. En effet considérons d'abord les triangles ayant 2 ou 3 sommets du type q_{N+1}, \dots, q_{N+4} . ils ne peuvent pas avoir de point q_1, \dots, q_N en leur intérieur par construction. Donc tout D est extérieur à chacun de ces triangles et on peut les retirer de la liste.

Soit maintenant un triangle ayant 1 sommets q_M du type q_{N+1}, \dots, q_{N+4} , un sommet q_L sur la frontière de D et un sommet interne q_k à D , s'il en existe. Formons le quadrilatère avec le triangle adjacent à $q_k q_M$. Soit q_P les 4eme point. Alors $q_P q_L$ devrait être une arête car elle vérifie le critère de Delaunay mieux que $q_k q_M$ si q_M est loin. Donc il ne peut y avoir de telle configuration.

Alors nécessairement, par la convexité de D un triangle ayant un sommet du type q_{N+1}, \dots, q_{N+4} a ses 2 autres sommets sur la frontière de D . Il est donc extérieur à D et peut être retiré de la liste.

5.3 Triangulation d'un domaine non convexe

Le domaine D est donné par des points destinés à devenir des sommets et ses arêtes frontières orientées avec la convention que le domaine est à gauche de sa frontière orientée.

5.3.1 Algorithme

- Trianguler d'abord l'enveloppe convexe du domaine par l'algorithme précédent.
- Pour toute arête frontière e de D qui ne serait pas une arête de la triangulation, trouver tous les triangles coupés par e . Ils forment un polygone P_e . Considerer P_e comme la reunion de 2 polygones, l'un P_{ed} à droite de e , l'autre P_{el} à gauche.
- Appliquer le présent algorithme récursivement à P_{el} et P_{ed} .
- Retirer tous les triangles ayant pour sommet un point du type q_{N+1}, \dots, q_{N+4} .

- Retirer tous les triangles adjacent par une arête et à droite de la frontière de D.

Pour construire les polygones P_e , le mieux est de partir d'une extrémité d'arête, de parcourir la liste des triangles ayant ce point pour sommet (cf le champ `supp` dans la classe `Vertex`), de calculer leurs intersection avec e et de prendre l'autre triangle adjacent par l'arête qui coupe e , etc jusqu'à ce qu'on arrive à l'autre extrémité de l'arête e .

5.3.2 Comment donner les points internes

Dans la pratique on applique d'abord l'algorithme précédent sans points internes. Toute arête a donc ses sommets sur le bord.

Ensuite on suppose que l'utilisateur a déterminé la densité de point par la longueur des arêtes du bord. Plus il donne un maillage fin à la frontière plus il souhaite un maillage fin dans cette zone.

On donne donc un poids a à chaque sommet frontière égal à la moyenne des longueurs des 2 arêtes du bord adjacentes à ce sommet. Ensuite pour toutes les arêtes on regarde si la demi-somme d des poids de leurs sommets est inférieure à leur longueur. Si oui on coupe l'arête en 2 par le milieu, on change les 2 triangles adjacents en 4 triangles et on affecte au nouveau sommet la demi somme d pour poids.

A chaque itération il faut réappliquer le critère de Delaunay. On recommence jusqu'à ce que toutes les arêtes aient une longueur inférieure à la demi somme des poids de leurs sommets.

5.4 Un premier module de triangulation automatique en C

```

/***** Programme test pour mshptg.c *****/
/***** necessite les fichiers mshptg.c et mshptg.h *****/
#include <stdio.h>
#include <math.h>
#include "mshptg2.h"

#define nbsmx 100 // nombre max de sommets (pour declaration des tableaux)
#define nbsdmx 1 // nombre max de sous-domaines
#define nbtmx 2*nbsmx // nombre max de triangles
#define nbamx nbsmx+nbsdmx // nombre d'ar^ete

long c[2*nbsmx]; // coordonne'es des sommets en entier (out)
long tri[4*nbsmx+2*nbsdmx]; // tableau de travail (out)

```

```

long ar^ete[2*nbamx]; //tableau des ar^etes qui doivent etre retenues (in)
long sd[2*nbsdmx]; // numero de chaque sous-domaines (in)
long nu[6*nbtmx]; //sommets dans les triangles (1..3)+tableau de travail
long reft[nbtmx]; // tableau de travail (out)
float cr[2*nbsmx]; // coord des sommets specifiees (in) et finaux (out)
float hh[nbsmx]; //hh[i]= taille souhaitee des ar^etes autour de q^i (in)
long nbs; // nb de points specifie's (in) et totaux (out)
long nba; // nb d'ar^ete (in)
long nbt; // nb de triangles (out)
long err; // indicateur d'erreur (out)
long nbsmxx; // nb maximum de sommets permis en tout (in)
long nbsd; // nb de sous-dimaines
FILE* ff; // fichier pour les sorties

void main()
{ /* exemple */
  int i;
  nbs=5; // nombre de sommets frontiere definis si facultatif enleve'
  nba=nbs; // nombre d'ar^ete definie

  cr[0]=0; cr[1]=0; // 4x-----x 3 Les point du bords (ici 5 points)
  cr[2]=1; cr[3]=0; // | | la geometrie correspond au dessin
  cr[4]=1; cr[5]=1; // \ x 5 | chaque point est rentrer par 2 coor
  cr[6]=0; cr[7]=1; // / | convention fortran pour les tableaux
  cr[8]=0.3; cr[9]=0.7; //1x-----x 2

  for(i=0; i<=nbs; i++)
  {
    ar^ete[2*i]=i+1; // le point de depart de l'ar^ete
    ar^ete[2*i+1]=i+2; //point d'arrive' de l'ar^ete (convention fortran)
  }
  ar^ete[2*nba-1]=1; // la derniere ar^ete qui ferme le profil

  for(i=0; i<=nbs; i++) hh[i]=0.1; // la taille des ar^etes souhaite'es
  hh[2]=0.01;
  hh[8]=0.01;

  sd[0]=1; // le numero du domaine
  nbsmxx=10; // nb max de sommets

```

```

nbsd=1;    // nb de sous-domaines

mshptg_(cr, hh, c, nu, &nbs, nbsmxx, tri, ar\^ete, nba, sd, nbsd, reft,
        &nbt, 0.25, 0.75, &err);

printf("err= %d nbs= %d nbt= %d\n", (int)err, (int)nbs, (int)nbt);

ff=fopen("ctriangle.dta", "w"); // ecriture dans le fichier ctriangle.dta

fprintf(ff, "%d %d\n", (int)nbs, (int)nbt);
for(i=0; i<2*nbs; i+=2)
    fprintf(ff, "%f %f %i\n", cr[i], cr[i+1], 0);
for(i=0; i<3*nbt; i+=3)
    fprintf(ff, "%d %d %d %d\n", (int)nu[i], (int)nu[i+1], (int)nu[i+2], 1);
fclose(ff);
}

```

Les fichiers mshptg.c et mshptg.h sont accessibles par ftp, ainsi que la version fortran mshptg.f dont elles sont dérivées par traduction automatique (f2c). L'auteur de ce module est F. Hecht (Frederic.Hecht@inria.fr).

EXERCICE 17 (*obligatoire*)

Tester le programme précédent. Rajouter un petit trou carré. Il faudra faire un makefile pour lier ce fichier avec mshptg.c et la bibliothèque mathématiques (lm). Le programme imprime une ligne à la console et il est important que $nt =$ donne un chiffre > 0 ce qui est le signe que le programme marche. Pour vérifier que la triangulation générée est la bonne on pourra utiliser le programme de conversion au format gnuplot écrit à la leçon 2.

REMARQUE 12

On comprend mieux comment freefem génère des maillages. L'ordre border sert à définir les sommets du ou des contours ainsi que la liste des arêtes. L'ordre buildmesh appelle le module mshptg.

6 LEÇON : Frontières courbes, triangulation automatique et adaptative

6.1 Un nouveau mailleur

Nous allons maintenant décrire un module de maillage automatique et adaptatif entièrement écrit en C++ par F. Hecht. La partie adaptative permet de demander le raffinement par rapport au (produits de) Hessien -matrice des dérivées secondes- d'une (ou de plusieurs) fonction donnée(s).

Le problème du maillage adaptatif est qu'on introduit des sommets supplémentaires sur les bords en cours de calcul. Lorsqu'on définit le bord par arête, on perd l'information géométrique sur le bords, par exemple s'il s'agissait d'un cercle. Les sommets supplémentaires ne pourront donc pas être ramenés sur le cercle mais resteront sur l'arête approchant le cercle. D'où une perte de précision. Dans ce nouveau mailleur les données géométriques sont conservées de sorte que les sommets frontières peuvent "glisser" sur la frontière.

La structure de donnée devra permettre de se souvenir partiellement de l'équation approchée de la frontière.

6.1.1 Fichier frontière

Voici un exemple de spécification de la frontière: un carré unité que le triangulateur ne fera que diviser en 2 triangles.

```
MeshVersion 0
Dimension 2
MaximalAngleOfCorner 360
Vertices 4
0. 0. 1
1. 0. 1
1. 1. 1
0. 1. 1
Edges 4
1 2 1
2 3 1
3 4 3
4 1 4
Corners 4
1 2 3 4
End
```

- Comme précédemment chaque morceaux de frontière est défini par une liste d'arête (polygone)
- Mais la frontière n'est pas le polygone mais la spline passant par les sommets et définie par des propriétés de tangence.
- Entre 2 sommets d'une arête du polygone, la frontière est alors un polynome de degré 1,2 ou 3 suivant que zero,une ou 2 tangentes sont spécifiées aux extrémitées de l'arête.
- En plus chaque sommet frontière peut être marqué comme étant un coin, soit automatiquement si l'angle entre les 2 arêtes frontières passant par ce sommet dépasse l'angle maximum autorisé, soit parce que le sommet est spécifié comme étant un coin.

Ce module de triangulation est identique au précédent si les options suivantes sont choisies:

- L'angle maximum pour un coin est 360 degré
- On ne spécifie pas la taille hVertex minimum d'arête autour d'un sommet de sorte que celle-ci est prise à sa valeur par défaut, la taille des arêtes adjacentes.
- Tous les sommets frontières donnés sont des coins.
- Les sous domaines sont spécifiés.

Dans l'exemple suivant un carré 10x10 (ref=1) est troué par un pentagone non convexe (ref=2). Le dernier sommet q9=(4.5,4.5) n'est pas spécifié "corner" de sorte que la spline passant par ce point n'est pas une droite mais une courbe du 2eme degré; la tangente à la courbe en ce point est prise égale à q8q5 car ce sont les 2 points qui entourent q9. Par ailleurs une taille minimale d'arête est spécifiée pour chaque sommet par "hVertices" de sorte que les arêtes spécifiées seront raffinées. Enfin 2 arêtes internes sont spécifiées qui divisent le domaine en 2 sous domaines, eux aussi spécifiés.

```
MeshVersion 0
Dimension 2
MaximalAngleOfCorner 360
Vertices 9
0. 0. 1
```

```

10. 0. 1
10. 10. 1
0. 10. 1
3. 3. 2
6. 3. 2
6. 6. 2
3. 6. 2
4.5 4.5 2
Edges 11
1 2 1
2 3 1
3 4 1
4 1 1
6 5 2
7 6 2
8 7 2
9 8 2
5 9 2
1 5 3
3 7 3
Corners 8
1 2 3 4 5 6 7 8
hVertices
2. 2. 2. 2. 1. 1. 1. 1. 1.
SubDomain 2
2 1 1 1
2 4 1 2
End

```

6.1.2 Mots clefs du fichier frontière

Les mots clefs "MeshVersion" et "Dimension" sont obligatoires. Ensuite les mots clefs suivants sont reconnus:

- Vertices NbOfVertices for(i=0 to NbOfVertices){ x y ref}
- Corners NbOfCorners for(i=0 to NbOfCorners){ Vertex }
- RequiredVertices NbOfRequiredVertices for(i=0 to NbOfRequiredVertices){ Vertex }
- hVertices for(i=0 to NbOfVertices){ hsize }

- NormalAtVertices NbOfNormalAtVertices for(i=0 to NbOfNormalAtVertices){ Vertex x y}
- Edges NbOfEdges for(i=0 to NbOfEdges){ VertexIn VertexOut ref}
- EdgesTangence NbOfEdgesTangence for(i=0 to NbOfEdgesTangence){ Edge VertexInEdge x y}
- SubDomain NbOfSubDomain for(i=0 to NbOfSubDomain){ type if(type==2) Edge Orientation ref}

où Orientation vaut +1 ou -1 suivant que le soudomaine est à gauche ou à droite de l'arête orienté.

6.1.3 Conversion du premier vers le second trianguleur

```
class frontiere {
public:
    float *xy;           // cordinates (for point i, x in 2*i , y in 2*i+1)
    int *ng;             // boundary number for each point
    int nbp;             // number of boundary points
    long *s;             // edges (edge i: beginning in 2*i, end in 2*i +1)
    long nbs;            // number of edges
    long nbsd;           // number of each connected component
    long *sd;            // an edge for each connected component
    float *hh;           // weight of points (to insert inner points)
    frontiere() {nbp = nbs = nbsd = 0; sd = new long[50];}
    int ajoute_point(float x, float y, int ng);
    void ajoute_segment(int n1, int n2);
    void save(const char* filename) const; //OP 97
};

void frontiere::save(const char * filename) const
{ int i;
  ofstream file(filename);
  assert(!file.fail());
  file << "MeshVersion 0 \r Dimension 2 \r MaximalAngleOfCorner 360"<< endl;
  file<<endl;

  file << "Vertices " << nbp << endl;
  for(i = 0; i< nbp; i++)
  file << xy[2*i] <<'\t'<< xy[2*i+1] <<'\t'<< ng[i] << endl;
```

```

file<<endl;

file << "Edges " << nbs << endl;
for(i = 0; i<nbs; i++)
file << s[2*i]+1 <<'\t'<< s[2*i+1]+1 <<'\t'<< ng[i] << endl;
file<<endl;

file << "SubDomain " << nbsd << endl;
for(i = 0; i<nbsd ; i++)
file << "2\t" << sd[2*i]+1 << "\t1\t"<< i+1 << endl;
file<<endl;

file << "Corners " << nbp << endl;
for(i = 0; i<nbp; i++)
file << i+1 << endl;
file<<endl;

file << "End" << endl;
file.close();
}

```

6.1.4 L'algorithme employé

Dans ce nouveau mailleur, d'autres techniques de maillage automatique sont utilisées principalement pour des raisons de vitesse de calcul.

D'abords le maillage initial n'est pas construit en plongeant le tout dans un grand rectangle mais en géant l'enveloppe convexe C de l'ensemble des points donnés.

- Les points étant donnés dans un ordre quelconque, au départ C est un triangle formé des 3 premiers points.
- Pour chaque nouveau point on teste s'il est à l'intérieur de C ou non. Ceci se fait triangle par triangle de C .
- S'il est à l'intérieur de C on divise le(s) triangle(s) en 3 (2 fois 2).
- Sinon on ajoute à C les triangles extérieurs obtenus en joignant ce nouveau point aux points du bords de C qui sont intérieur au nouveau convexe obtenu avec ce nouveau point et C .

Pour calculer rapidement si un point est intérieur à un triangle on utilise la technique des Quadrees.

Enfin pour forcer les arêtes du bords à être des arêtes du maillage et pouvoir ainsi retirer les triangles inutiles on utilise l'algorithme aléatoire de Bourouchaki:

- Faire une boucle sur les arêtes e du bords qui ne sont pas des arêtes du maillage.
- Pour chaque e , trouver l'ensemble des arêtes du maillage qui coupent e . (Partir d'une extrémité de e et utiliser les triangles ayant ce point pour sommet pour trouver la première arête, puis marcher en utilisant le triangle à gauche ou à droite de cette arête et ainsi de suite.)
- Pour chaque quadrilatère formé par les 2 triangles adjacents à une arête qui coupe e échanger aléatoirement de diagonale jusqu'à ce qu'aucune des ces arêtes ne coupent e .

L'algorithme converge nécessairement car l'aléatoire explore nécessairement toutes les configurations or on sait que les configurations viables existent.

6.2 Compléments

6.2.1 Polygones de Voronoi

Soit $Q=(q_1,q_2,\dots,q_N)$ N points. A chaque point q_i on associe le polygone de Voronoi V_i dont la définition est

$$V_i = \{q = (x, y) : \|q - q^i\| \leq \|q - q^j\|, \quad \forall j \neq i\}.$$

Il est facile de voir que les frontières de V_i sont des portions de bissectrices entre les points q_i et q_j .

Il est aussi immédiat de voir que les arêtes de la triangulation de Delaunay sont les segments $q_i q_j$ pour tout q_j dont une portion de bissectrice à q_j - q_i fait partie d'une des frontières de V_i .

Ainsi on voit que la triangulation de Delaunay est unique dès qu'aucun quadruple de points ne sont pas sur un même cercle.

6.2.2 Quadtree

Considérons un point (x,y) du carré unité. On divise le carré en 4 petits carrés égaux et on teste dans lequel est le point (x,y) . S'il est dans celui de

gauche en bas on assigne au point le couple $(0,0)$, s'il est en haut à droite on assigne $(1,1)$, s'il est en haut à gauche $(0,1)$ et sinon $(1,0)$. On redivise le petit carré en 4 et on fait la même opération. On obtient ainsi 2 couples de nombres binaires associés au point et on les concatène. Si le point est en haut à droite dans la première division et en haut à gauche dans la deuxième il aura alors l'étiquette $(10,11)$. Et ainsi de suite jusqu'à ce qu'on arrive à la précision maximal des entiers. On a construit ce qu'on appelle un arbre quaternaire (quadtrees), produit de 2 arbres binaires.

Supposons qu'on ait un triangle (q_0, q_1, q_2) dans le carré unité et qu'on veuille savoir si un point q appartient au triangle. On peut dégrossir fortement le problème en utilisant une représentation des points par quadtrees. On cherche le carré quadtrees qui contient les 3 sommets. Pour cela il suffit de comparer bit à bit la représentation quadtrees des points et de garder la partie commune. On cherche enfin si q est dans ce carré. S'il ne l'est pas q ne peut être dans le triangle. Pour savoir si le point est dans ce carré il suffit de vérifier que le début de sa représentation quadtrees est bien celle du carré.

Dans le cas général, il faut donc changer les échelles pour se ramener au cas du carré unité. On aura aussi intérêt à convertir toutes les coordonnées en coordonnées entières (par changement d'échelle) de manière à supprimer les indéterminations dues à la limite de précision des représentations flottantes.

Inversement, si la question est dans l'autre sens: soit un point q dans un domaine triangulé, dans quel triangle est t'il? avec le quadtrees, on va trouver très rapidement une cellule, l'un des sommets du maillage le proche du point q , puis après on se promène par adjacence dans le maillage comme suit: calcule des 3 aires signées dans une triangles t de sommet s_1, s_2, s_3 :

$$a_{12} = \text{aire}(s_1, s_2, q) \text{ arête } 12,$$

$$a_{13} = \text{aire}(s_1, q, s_3) \text{ arête } 13,$$

$$a_{23} = \text{aire}(q, s_2, s_3) \text{ arête } 23,$$

si les 3 aires sont positives on a gagné. si l'une est négative on prend le triangle adjacent à l'arête si 2 sont négatives on prend le triangle adjacent à l'arête au hasard qui a une aire négative ; si les 3 sont négatives alors bug.

6.3 Maillage adaptatif

Ce nouveau maillage peut adapter une triangulation à une ou plusieurs fonctions données. Le principe consiste à changer la métrique qui définit la distance dans l'espace.

Soit M une matrice symétrique définie positive. On définit une nouvelle norme et une nouvelle distance

$$\|x\|^2 = x^T M x, \quad d(x, y) = \|x - y\|$$

Le critère de Delaunay sera désormais calculé avec cette distance de sorte que les cercles deviennent des ellipses. Toute la théorie et les algorithmes exposés plus haut marchent encore avec ce nouveau critère de Delaunay elliptique. La distance apparaît aussi à une autre étape de l'algorithme: lorsqu'on divise les arêtes trop longues.

Pour construire M on se donne une fonction $u(x, y)$ et on prend pour M la matrice des dérivées secondes de u .

On rappelle que l'erreur entre l'interpolé, u_h , de u et u est majorée par

$$\|u - u_h\|_{E,1} \leq c \|u''\|_{E,\infty} h$$

où E indique qu'il s'agit de la norme Euclidienne classique pour les normes de H^1 et de L_∞ .

On voit ainsi que si u'' est grand alors l'erreur est grande, d'où l'idée de maintenir $u'' h$ constant. Il faudrait une estimation plus fine pour voir que les directions jouent aussi le même rôle et que si une composante de u'' est grande alors l'erreur dans cette direction est grande aussi.

De même si on souhaite adapter le maillage par rapport à plusieurs fonctions, par exemple u et v , on considèrera la matrice produit $u'' v''$ symétrisée.

Remarquez toutefois que rien ne garantit que cette matrice ait des valeurs propres strictement positives, il faudra donc se protéger contre ce problème, par exemple en prenant

$$M = Q^T |\Lambda| Q, \quad |\Lambda| = \begin{pmatrix} |\lambda_1| + \epsilon & 0 \\ 0 & |\lambda_2| + \epsilon \end{pmatrix}$$

où Q est la matrice de rotation qui rend u'' diagonale et où on a pris la valeur absolue des valeurs propres plus un petit nombre pour rendre la matrice définie positive.

Dans la pratique on souhaite souvent optimiser le maillage après coup; c'est à dire qu'on calcule d'abord une approximation grossière de la solution du problème (une EDP dans notre cas) et ensuite on souhaite optimiser le maillage pour obtenir une meilleure solution. Dans ce cas la fonction u n'est pas donnée analytiquement mais par ses valeurs sur le maillage initial. Il est donc plus approprié de supposer qu'en entrée (input) on se donne un "maillage de fond" et une ou plusieurs fonctions P1 sur ce maillage dont le Hessien (dérivées secondes) servira à définir la nouvelle distance.

REMARQUE 13 Absence de Métrique spécifiée

Si l'utilisateur ne spécifie pas la métrique dans le nouvel algorithme alors on utilise par défaut la matrice identité multipliée par $h(x,y)$, la fonction P1 interpolée des poids aux sommets définis par la longueur des arêtes du maillage frontière comme précédemment.

6.3.1 Fichier pour la triangulation et le maillage adaptatif

Il faut 2 fichiers: un fichier maillage identique à celui généré par le programme de maillage écrit plus haut et qui s'appelle ici "in.mesh" et un fichier pour définir la métrique: "in.mtr" selon l'un des 3 formats:

-0. Sans maillage adaptatif

On donnera alors seulement un fichier "geom.in". Il faut que les autres fichiers définis plus bas n'existent pas. (Sous unix exécuter avec l'option -h pour avoir la liste des paramètres). Le fichier généré en sortie s'appelle "out.msh".

Dans les 2 options adaptatives geom.in est aussi utilisé et on ne peut pas changer le nom de ce fichier car celui-ci est écrit dans out.msh.

-1. Adaptation isotropique:

```
nv 1
h_1
...
h_nv
```

ou nv est le nombre de sommets et hi est le pas souhaité autour du sommet i.

-2. Adaptation anisotropique

```
nv 3
a11_1 a12_1 a22_1
...
a11_nv a12_nv a22_nv
```

ou nv est le nombre de sommets et a_{ij-k} est la métrique souhaitée autour du sommet k.

6.4 Renumerotations

Si l'on souhaite résoudre les systèmes linéaires par une méthode de Gauss il est indispensable de renuméroter les noeuds et éventuellement les éléments d'un maillage (méthode frontale).

De nombreuses méthodes existent pour effectuer ce type d'opérations. Nous allons décrire en détails la méthode de Gibbs qui est entièrement automatique et nous mentionnerons très brièvement d'autres approches.

Afin de pouvoir décrire cet algorithme, nous allons introduire (ou rappeler) les notations suivantes:

- \mathcal{T}_h désigne la triangulation,
- T_j l'élément j de ce maillage,
- n_t le nombre d'éléments de \mathcal{T}_h ,
- n_v le nombre de noeuds de \mathcal{T}_h .

La renumérotation des noeuds, selon la méthode de Gibbs, comprend formellement trois étapes consistant en:

- la recherche d'un bon point de départ,
- l'optimisation de la descendance de la numérotation,
- la numérotation avec l'algorithme de Cuthill MacKee, numérotation éventuellement renversée.

Nous allons maintenant détailler ces trois phases; en premier lieu nous définissons les notations supplémentaires suivantes:

- x, y, \dots sont les noeuds de la triangulation \mathcal{T}_h ,
- y est voisin de x s'il existe un élément T de \mathcal{T}_h tel que x et y en soient des noeuds,
- le degré d'un noeud x est le nombre de ses voisins, il est noté $d(x)$,
- les voisins de x constituent $N_1(x)$ le niveau 1 de sa descendance,
- les voisins des noeuds de $N_1(x)$ non encore répertoriés forment $N_2(x)$, le niveau 2 de la descendance de x (un noeud apparaît au plus une fois dans $N_k(x)$ et par suite n_t peut être dans les autres niveaux),
- l'ensemble des niveaux $N_k(x)$ constitue la descendance, notée $D(x)$ du noeud x ,
- la profondeur p de la descendance est le nombre de niveaux de celle-ci,

- le graphe G des noeuds représente les connexions entre les noeuds (en terme de voisins), la figure donne les graphes relatifs à un triangle et à un quadrangle,
- le graphe G contient une ou plusieurs composantes connexes notées C_k .

Les trois phases consistent alors en:

- **La recherche d'un bon point de départ :**

Parmi les noeuds du contour du domaine, on sélectionne le noeud x tel que $d(x)$ soit minimal.

Soient $D(x)$ sa descendance et $N_p(x)$ son dernier niveau; on réalise alors l'algorithme suivant:

- Début
- Pour tous les noeuds y de $N_p(x)$, dans l'ordre croissant de leur degré $d(y)$, on calcule $D(y)$ et sa profondeur $p(y)$,
- Si $p(y)$ est supérieur à $p(x)$ alors y est sélectionné, on fait $x=y$ et on va à Début,
- Sinon on teste un nouvel y .

Le résultat nous donne un noeud y , extrémité d'un pseudo-diamètre dont l'autre extrémité x est un point du dernier niveau de la descendance de y tel que son degré soit minimal.

La phase deux de la méthode est alors entreprise:

- **L'optimisation de la descendance de la numérotation :** Soient:

x et $D(x) = (N_1(x), \dots, N_p(x))$

y et $D(y) = \{N_1(y), \dots, N_p(y)\}$

les descendance des points du pseudo-diamètre; on cherche à construire $D = (N_1, \dots, N_p)$ qui nous donnera la nouvelle numérotation des noeuds. Ce processus se décompose en plusieurs phases:

- Boucle de $w=1$ à n_v

Construction des couples (i,j) associés à w tels que:

(i,j) est formé si w est dans $N_i(x)$ et w dans $N_{(p+1-j)}(y)$

Fin de boucle.

- A partir de ces couples (i,j) , on commence à construire les N_i de la manière suivante:
 Boucle de $w=1$ à n_v
 Si (i,j) associé à w existe:
 mettre w dans N_i et retirer w du graphe G des noeuds
 Fin de boucle.

Si $G = 0$ aller à l'étape finale (Numérotation selon C.McK.)

Sinon on classe selon leur cardinal les t composantes connexes C_k du graphe G et on cherche à répartir les noeuds restants dans les niveaux N_i comme suit:

- Boucle de $k=1$ à t
- Boucle de $m=1$ à p , les niveaux de descendance
 on calcule:
 $nm = \text{card } N_m$
 $hm = nm + \text{card}(\text{noeuds de } C_k \text{ dans } N_m(x))$
 $lm = nm + \text{card}(\text{noeuds de } C_k \text{ dans } N_m(y))$
 Fin de boucle.

Soient $h_0 = \max(m=1..p)(hm \text{ tel que } hm-nm \text{ positif})$ et
 $l_0 = \max(m=1..p)(lm \text{ tel que } lm-nm \text{ positif})$

- si h_0 est plus petit que l_0 alors: placer chaque noeud de la composante C_k dans le niveau N_i défini par le numéro i du couple (i,j)
- sinon si l_0 est plus petit que h_0 et si $\max(i=1..p)(\text{card } N_i(x))$ est plus grand que $\max(i)(\text{card } N(y))$ ranger i , sinon ranger j .

Fin de boucle.

Ainsi une bonne répartition des noeuds est obtenue dans les différents niveaux.

- L'algorithme de Cuthill MacKee est alors utilisable à partir de ces résultats.
- **La numérotation selon l'algorithme de Cuthill MacKee**, éventuellement renversée:

Si $d(y)$ est plus petit que $d(x)$, on échange x et y et on renverse les niveaux associés:

$$N(p-i+1) = N_i \text{ pour } i=1\dots p$$

Notons $NEW(x)$ le nouveau numéro correspondant à l'ancienne valeur n du noeud x . On pose:

$$NEW(x) = 1$$

$$\text{Initialisation: } n=1 \text{ et } N_0 = (x)$$

On boucle alors sur les p niveaux de la descendance:

- Boucle de $k=0$ à $p-1$
 - a) tant qu'il existe w plus petit numéro des noeuds de N_k précédant des voisins non renumérotés,
faire: $C(w) = \text{voisins}(w)$ intersecté avec $N(k+1)$
 - b) tant qu'il existe s de degré minimal de $C(w)$ non renuméroté,
faire: $n = n+1$ et $NEW(s) = n$
- Si a) échange x et y et b) choisit $N_i(x)$ pour C_1
- ou si a) n'échange pas x et y et b) choisit $N_i(x)$ pour C_1

Fin de boucle.

Alors :

- Boucle de $i=1$ à nv
 $s = NEW(i)$
 $NEW(i) = (nv+1 - s)$
Fin de boucle

c'est le renversement de la renumérotation dont l'intérêt est dû au résultat suivant: la largeur du profil est globalement meilleure ou au moins aussi bonne que la largeur avant renversement.

Cet algorithme est très performant pour réduire le profil d'une matrice, c'est-à-dire pour minimiser la différence entre les numéros des noeuds d'un même élément.

Pour illustrer cette propriété, nous allons donner quelques exemples. Nous introduisons les définitions suivantes:

Le *profil total* est la somme du nombre de colonnes (resp. lignes) comprises entre la première colonne (resp. ligne) non vide et la diagonale (dans le cas d'une matrice symétrique).

Le *profil moyen* est le quotient de cette valeur par le nombre de colonnes (lignes).

La *largeur* (ou largeur de bande) est le maximum des écarts entre la première colonne (ligne) remplie et la diagonale (dans le cas symétrique encore une fois). Cette largeur est la quantité que l'on cherche à minimiser dans le cas des méthodes de résolution directe.

Maillage	1	2	3	4	5	6
Nombre de noeuds	229	908	1559	61	3696	2056
Profil total initial	19592	42564	604714	1024	2266436	373601
Profil moyen initial	85.55	46.87	387.8	16.7	613.21	181.713
Largeur initiale	217	897	1536	57	1183	2036
Profil total final	2637	25844	64931	379	436632	284920
Profil moyen final	11.51	28.46	41.64	6.21	118.13	138.580
Largeur finale	18	45	66	13	233	225

Tableau 13.1 : Profil initial et profil après renumérotation.

Maillage	6	7
Nombre de noeuds	2056	2056
Profil total initial	373601	284920
Profil moyen initial	181.713	138.580
Largeur initiale	2036	225
Profil total final	284920	284883
Profil moyen final	138.580	138.562
Largeur finale	225	225

Tableau : Renumerotation

effectuée sur un maillage déjà renuméroté.

6.4.1 Une fonction de renumérotation

On trouvera dans le fichier `gibbs.cxx` de `freefem` un implémentation de la méthode décrite. La fonction prend un maillage comme entrée, ressort le même maillage renuméroté en sortie et renvoie un message d'erreur éventuel:

```
typedef struct { float x,y; }rpoint;
typedef long triangle[3];

typedef struct
{
  long np, nt;
  rpoint *rp;
  triangle *tr;
  int *ngt, *ng;
```

```

} triangulation;

int renum (triangulation * mesh)

```

EXERCICE 18

Identifier dans la façon dont renum appelle gibbs ou est le tableau qui donne le nouveau numéro d'un point en fonction de l'ancien.

EXERCICE 19 (*Obligatoire*)

Remplacer l'ancien mailleur par le nouveau dans l'interpreteur DBI2. (Contacter Pironneau par email pour avoir les fichiers sources si vous ne les avez pas. Ils seront disponibles ultérieurement dans ftp://ftp.ann.jussieu.fr/pironneau.) les fichiers sources contiennent: l'interpreteur DBI2 et le mailleur MFH. Ces deux modules fonctionnent de manière indépendante, c'est à dire qu'il faut d'abord lancer DBI2, ce qui produit un fichier "prepmesh.dta" lequel sert de fichier données pour MFH. DBI2 contient aussi toujours l'ancien mailleur. Le but de cet exercice est d'intégrer MFH à l'intérieur de DBI2 et donc de retirer l'ancien mailleur. Pour plus de détail voir la description du projet ci-dessous.

Aide Procedure pour incorporer le nouveau mailleur (dossier meshpp2) dans le nouvel interpreteur (dossier NDBI2).

- 1. Copier tous les fichiers de meshpp2 dans le dossier NDBI2
- 2. remplacer le makefile par ceci (ibm rs6000)

```

#nouveau makefile
ndbi2= Analyse.o TestAnal.o ugraph2.o xrgraph.o Fonction.o triangl2.o gibbs2.o
Mesh2.o Xrgraph.o QuadTree.o io.o ppm.o Metrix.o

nbp: TestAnal.o $(ndbi2)
        x1C -o dbi TestAnal.o $(ndbi2) -lX11 -lm

#fin de fichier

```

Ce makefile est compact mais on perd la compilation séparée. (Si par exemple vous changer ppm.C il faudra faire (sur ibm rs6000 ou le comilo s'appelle x1C)

Note: ne pas allumer l'option -DDRAWING, prévue dans le mailleur pour la mise au point car alors il y aura un conflit d'appel à "init-graphique()" entre DBI et mesh++ en sortie de ce dernier.

```
x1C -c ppm.C
rm dbi
make
```

- 3. Dans ppm.C changer le main en une fonction "maketriaui" en changeant la ligne main(... en

```
int maketriaui(int argv, char** argv)
```

- 4. Creez un fichier "ppm.h" contenant seulement la ligne

```
int maketriaui(int, char**);
```

- 5. Dans Analyse.C au debut rajouter

```
#include "ppm.h"
```

Commentez la ligne

```
err = MakeTriangulation(...
```

et rajoutez apres la ligne suivante (save...) la ligne

```
char** junk; err = maketriaui(1,junk);
```

- 6. Recuperez les donnees en sortie de maketriaui qui se trouve dans la variable Th et recopiez les dans la variable t. La variable t est du type triangulation et la classe triangulation est definie dans le fichier "triangul.h" alors que Th est du type triangles et la classe triangles est definie dans "mesh2.h". Il faut faire attention a dimensionner correctement les tableau avant de les recopier.

L'équivalence des champs est comme suit

classe triangulation	classe Triangles
-----	-----
for(int i=0;i<Th.nbt;i++)	//Th conserve des triangles inutiles
if(Th.triangles[i].link) t.nt++;	Th.nbt
t.np	Th.nbv
t.rp[i].x	Th.vertices[i].r.x
t.rp[i].y	Th.vertices[i].r.y
t.ng[i]	Th.vertices[i].ref
t.me[i][j]	Th.Number(Th.triangles[i][j]) (tester link)

Attention, il faut aussi tester $\text{Th.triangles}[i].\text{det} > 0$ sinon on recupère des triangles ayant un point a l'infini. Enfin il faut aussi éliminer les triangles qui sont dans les trous du domaines.

EXERCICE 20

Rajouter dans le programme issu de l'exercice précédent l'appel à la fonction `gibbs` pour la renumérotation.

EXERCICE 21

Rajouter dans l'interpréteur le mot réservé "savemesh" avec sa fonctionnalité c'est à dire par exemple: `savemesh("test.msh");`

Pour faire cela on consultera la leçon suivante ainsi que l'aide ci-dessous.

6.4.2 lecture de chaine de caratères dans DBI

- Ajouter un membre privé dans la classe `Analyseur`: `char* curString;`
- Ajouter dans la liste des symboles: `chaîne`
- Modifier `Analyseur::NextSym()` en incluant un case `""`: et ici copier les caractères lus dans un buf jusqu'a avoir trouvé l'autre `""` (a moins que le buffer soit plein). Voir qq lignes plus haut `if(isalpha(i)...` Puis faire `cursym = chaîne; curString = new char[n+1]; strcpy(... break`
- Enfin là ou il faut reconnaitre une chaîne on fait `fname = curString; match(chaîne);`

6.4.3 Bug dans DBI

Il faut qu'il y ait une correspondance entre la liste "Symbol" et le tableau `SymbolName`:

```
static char *SymbolName[] = {
    "(", ")", "{", "}", "constante", "identificateur", "fonction", "+",
    "-", "*", "/", "^", "<", "<=", ">", ">=", "=", "!", " ", ";",
    "&&", "||", "if", "then", "else", "erreur", "fin", "=", "buildmesh",
    "border", "fonction2", "derive", "plot", "string", "savemesh"};
```

Il faut corriger la dernière ligne conformément à ce qui est ci-dessus.

7 Projet

7.1 Orientation

L'objectif du projet est de participer activement à la construction de freefem 4.0. La version précédente ayant grandit trop vite, à une période de transition entre le C et le C++, le source n'est plus présentable. Nous en profiterons aussi pour corriger certains défauts, comme l'absence d'interface pour les arêtes par exemple, et pour rajouter de nouvelles fonctionnalités comme la différentiation symbolique.

7.2 L'objectif

L'objectif n'est pas de réécrire freefem entièrement mais d'obtenir un outil de base pour préparer les données pour le solveur écrit plus haut. On rappelle que le solveur résout

$$au - \nabla \cdot (k\nabla u) = f, \text{ dans } \Omega,$$
$$u = g \text{ ou } k\frac{\partial u}{\partial n} + bu = h \text{ sur } \partial\Omega$$

La distinction entre les 2 conditions aux limites se faisant par la convention que si g est non nul en un sommet alors il y aura une condition de dirichlet en ce point. Par ailleurs on prolonge par zero b et h sur toute la frontière et on résoud le problème

$$\min_{u \in V} \int_{\Omega} (au^2 + k|\nabla u|^2 - 2fu) + \int_{\partial\Omega} (bu^2 - 2hu)$$

où V contient les conditions de Dirichlet.

Ainsi le module de résolution a besoin de

- une triangulation,
- les fonctions a,b,k,h,f,g.

Bien qu'on demande d'insérer les sources du solveur dans le même makefile que le reste, on ne reliera pas, dans un premier temps l'interpreteur/mailleur avec le solveur; ils communiqueront par fichiers, c'est à dire que

- le solveur lit les fichiers triangulations et les fichiers fonctions
- la visulation du résultat du solveur est faite par la fonction equipot immédiatement en sortie du solveur.

- la fonction solution du solveur n'est pas réutilisable ensuite puisque ce n'est pas une fonction Gfem comme les autres mais un tableau de valeur.

On pourra éventuellement ajouter dans DBI2 la relecture du résultat du solveur et faire une étape d'adaptation du maillage.

7.3 Les étapes

Il faut travailler par étape: terminer une étape, l'archiver avant de démarrer la suivante. Ceci de manière à éviter de tout perdre et aussi pour pouvoir présenter à tout moment l'état du projet et éviter de répondre "ça tourne presque".

- A. Faire fonctionner la boucle Maillage + donnees + EDP (avec $b=0$) + dessin du résultat par equpot en utilisant DBI2 et votre solveur intégré dans un seul exécutable. Ceci correspond à l'exercice de la section précédente.

Il faudra construire un ou plusieurs fichiers contenant les valeurs de a, b, k, h, f, g . Le solveur lira alors ces fichiers ainsi que le fichier de la triangulation et produira un dessin des courbes de niveau et un fichier résultat.

Attention la syntaxe de Gfem a légèrement changée. Par exemple:

```

pi = 4*atan(1);
buildmesh(x,y,ref,1000)
{
border(t=0,pi*2,50)
{
ref = 1;
x = cos(t);
y = sin(t);
};
border(t=0,2*pi,20)
{
ref = 2;
x = 0.3+0.3*cos(-t);
y = 0.3*sin(-t);
};
};

```

```

fonction g1(w) g1 = 0.2*sin(x)*sin(x) + 0.02*y*y + 0.02*w;
adaptmesh(g1(1));
savemesh("test.msh");

```

```

fonction g(w) g = (x*x+y*y+ 1.0e-10)*(ref>=2) ;
fonction2 f(q,p) f = -4;
fonction h(w) h = 2;
    save("f.dta",f(q,p));
    save("g.dta",g(1));
    save("h.dta",h(w));
solve(u);

```

L'implémentation de "save", "savemesh", "adaptmesh" et de "solve" n'étant pas faite, il faudra les faire d'abord.

Ici adaptmesh opère très simplement en générant un fichier de valeurs à partir de g1 avec w=1. Ce fichier est un .mtr pour le trianguleur (voir plus bas).

Tout ces étapes sont obligatoires. A partir de maintenant il s'agit d'option complémentaires et facultatives.

- C. Faire le cas b non nul.
- D. Incorporer le solveur à l'intérieur du programme en définissant le mot clef "solve" avec par exemple la syntaxe

```

solve(u;a,b,k,h,f,g);

```

- E. Test de " adaptmesh(u)" (voir plus bas).
- F. Casting de la solution u au format des fonction Gfem (voir plus bas).
- G. Suppression des mots clefs "fonction, fonction2".
- H. Etude de sensibilité

Le langage permettant le calcul formel de dérivée, on peut étudier la sensibilité par rapport à un paramètre. Par exemple, si le second membre est une fonction de t, $f(t,x,y)$ définie par

```

fonction f(t) { f = x*(y+t^2) };

```

alors on pourra obtenir la dérivé en t de la solution de l'edp en changeant dans le problème f par fp défini par

```
fonction fp = deriv(f,t);
```

- H. Introduire la notion de spline pour les frontières, soit par exemple:

```
spline
{ ref=1;
  vertexlist
  {
    0.1 0.2 normal(1,2)
    1.2 3.4 corner
    1.4 3.4 nocorner
    1.5 3.8
  };
};
```

7.4 Aide à la réalisation du projet

7.4.1 L'instruction "save"

Suivre la syntaxe "save(filename,expression)". On s'inspirera alors de "plot" et de "savemesh".

7.4.2 L'instruction "solve"

Au début on fera très simple, par exemple "solve(u)". Il faudra donc rajouter l'instruction solve, mais l'argument, ici "u", est une variable et non une expression. Il faudra vérifier que s'il elle n'existe pas elle soit créée. Mais dans un premier temps on peut communiquer les données au solveur entièrement par fichier. Ainsi le solveur, ex dans le cas d'un Laplacien, sera une fonction sans paramètre (ex: void solvepde()) qui lira toutes ses données, la triangulation dans un fichier "test.msh", le second membre dans "f.dta", les conditions de Dirichlet dans "g.dta" et de Neumann dans "h.dta". Cela donne le programme Gfem suivant:

```
{
pi = 4*atan(1);
buildmesh(x,y,ref,1000)
{
border(t=0,pi*2,50)
```

```

{
ref = 1;
x = cos(t);
y = sin(t);
};
border(t=0,2*pi,20)
{
  ref = 2;
  x = 0.3+0.3*cos(-t);
  y = 0.3*sin(-t);
};
};
savemesh("test.msh");
fonction2 f(u,v) f = -4;
fonction g(w) g = (x*x+y*y)*(ref>=2);
fonction h(w) h = 2;
plot(g(1));
  save("f.dta",f(u,v));
  save("g.dta",g(w));
  save("h.dta",h(w));
solve(p);
}

```

On affichera les résultats à l'intérieur de la fonction `Isolve::execute()` de sorte que le nom "p",ici, n'intervient pas.

Dans un deuxième temps, et là il y a une difficulté réelle, on ajoutera une structure pour pouvoir utiliser la fonction `p` dans la suite du programme, par exemple si en fin de programme `Gfem` on avait `plot(2*p-1)`. Le problème réside dans le fait que la variable `p` est en fait une fonction mais il lui manque la fonction "eval()".

7.4.3 L'instruction "adaptmesh"

Comment calculer une approximation de la dérivée second d'une fonction P1?

calculer $dx(u)$ et $dy(u)$, ceux-ci sont constants sur les triangle,

Calculer ensuite une interpolation P1 de ces deux fonction par la formule

$$dxu(i) = \text{int}(dx(u)w^i)/\text{int}(w^i)$$

$$dyu(i) = \text{int}(dy(u)w^i)/\text{int}(w^i)$$

Notez que toutes les integrales sont locales et se calculent toutes en meme temps par assemblage sur les triangles.

Recommencer sur dxx(u) en derivant dxu puis reinterpoler pareillement

Pour la syntaxe, pas de problème: "adaptmesh(expression)". Donc l'implémentation dans DBI se fait comme pour plot. On pourra générer un fichier "in.mtr" pour stocker les valeur de la metrique suivant le format:

```
nb_vertex 1
metric[0]
...
metric[nb_vertex-1]
```

Le 1 pouvant devenir un 3 si on souhaite avoir des metriques non-isotropes.

Par ailleurs il faudra avoir 2 options d'appel de la fonction de triangulation: l'une sans maillage adaptatif qui ne prend comme entrée que le fichier géométrie et une option avec maillage adaptatif qui prend pour entrées un fichier maillage (ex mesh.mesh), un fichier de métrique (ex in.mtr) et le fichier geometrie (ex in.geom) Donc il faut modifier ppm.C par exemple en jouant sur la valeur de argc.

```
fmeshback="mesh.msh"; // background mesh
fmeshout="mesh.msh";
fmetrix="in.mtr"; // background metric
if(argc==1) { fmeshback=NULL; fmetrix = NULL;} //OP
```

7.4.4 Casting de u au format DBI

Evidement il est beaucoup plus difficile de faire "adaptmesh(u)" ou u est un résultat du solveur car il n'est pas au format des fonctions de DBI.

L'idée de base est qu'il faut neutraliser l'évaluation de u(x,y) comme une fonction2 et qu'il faut utiliser à la place par exemple

```
float operator()(float x, float y) { return interpolate(t,val,x,y);}
```

où val est les tableau de valeurs sur la triangulation t.

On définit donc une classe dérivée de CVirt2:

```
class CTab: public CVirt2 { // fonction tableau
public:
triangulation *t;
```

```

float* val;          // les valeurs sur la triangulation
CTab(triangulation* tt, float* vval) : t(tt), val(vval) {}
float operator()(float x, float y) { return interpolate(t,val,x,y);}
float valeur(int i) { return val[i];}
};

```

et on rajoute un champs

```
CTab *f2;
```

dans la classe Isolve qui sert à définir l'instruction solve.

La lecture de solve(u) se fait comme suit:

```

case solve:
  nextSym();
  match(lpar);
  match(iden);
  if (curIden->type != Iden::inconnu) erreur("Can't redefine a function");
  curIden->type = Iden::fonction2;
  CTab *f = new CTab (&t, 0); // On cr\`ee la fonction
  curIden->f2 = f;
  res = new Isolve(f,an.x->storage, an.y->storage, an.ng->storage);
  match(rpar);
break;

```

A l'exécution de Isolve on aura

```

void Isolve::execute()
{
  solvepde();// Resout l'EDP. Toutes les donn\`ees sont dans des fichiers.
  float* resultat = new float[t.np];
  ... // lecture du fichier "sol.dta"
  f2->val = resultat;
  equipot(t.ng,temp,20, 2); // trace la solution ancien style.
}

```

Pour un tracé "nouveau style" de la solution par un ordre Gfem, on ne peut pas faire "plot(u(0.0,0.0))" car ceci donne des valeurs à x,y et d'un autre côté on ne peut pas appeler u sans paramètre; on rappelle que les fonctions sans paramètres formels ne sont pas définis dans Gfem, ce qui est évidemment une lacune. On rappelle aussi que toutes les fonctions sont implicitement fonction de x,y,ref, après que buildmesh ait été effectué.

Un pis-aller consiste à faire

```
... solve(u);  
fonction uu(w) uu = u(x,y);  
plot(uu(1));
```

8 LEÇON: L'interpreteur DBI

[Cliquer ici](#)

References

- [1] G. Buzzi-Ferrari: Scientific C++ Addison-Wesley 1993
- [2] A. Koenig (ed.) : Draft Proposed International Standard for Information Systems - Programming Language C++. ATT report X3J16/95-087 (ark@research.att.com)
- [3] B. Lucquin, O. Pironneau: Introduction au calcul scientifique. Masson (1996).
- [4] J. Shapiro: A C++ Toolkit, Prentice Hall 1991.
- [5] B. Stroustrup: The C++ Programming Language Addison Wesley 1991