

Numerical Methods for Partial Differential Equations

Olivier Pironneau

October 23, 1998

Contents

0.1	Forewords	1
0.2	Dates	2
0.3	Why go on the web?	2
0.4	Diploma	2
0.5	Who correct the exercises?	2
0.6	The content	2
0.7	What you need to do.	3
1	LESSON: Introduction	3
1.1	Orientation	3
1.2	How to read a file	3
1.3	Gnuplot	7
1.4	batch file	8
2	LESSON on freefem and gnuplot	8
2.1	Orientation	8
2.2	Problem statement	9
2.3	An example with freefem	9
2.4	Triangulation file	10
2.5	Read a Triangulation file	11
2.6	Display a Triangulation with gnuplot	13
2.7	Vizualisation of a function	15
2.8	Complement: the debugger gdb	16
2.8.1	The command where	17
2.8.2	Parameter passing in UNIX	17
2.9	Display of the level lines of a function	18

2.9.1	Principle: P1 interpolation	18
2.9.2	Error estimates	20
3	Lesson: An explicit scheme for the wave eq.	20
3.1	The problem	20
3.2	The scheme	20
3.3	The program	22
3.4	Exercise	23
4	Lesson: An implicit scheme	24
4.1	The Crank-Nicolson implicit scheme	24
4.2	Discretization in space	24
4.3	The conjugate gradient algorithm	27
4.3.1	Remark for non zero Dirichlet conditions	27
5	Lesson: Automatic triangulation	28
5.1	Automatic triangulation of a square	28
5.1.1	The problem	28
5.1.2	Algorithm	28
5.1.3	The Delaunay criteria	29
5.2	Triangulation of a convex domain	29
5.2.1	Algorithm	30
5.3	Triangulation of a non convex domain	30
5.3.1	Algorithm	30
5.3.2	Generating interior points	31
5.4	Un premier module de triangulation automatique en C	31
5.5	Mesh Adaption	33
5.6	A Fast Finite Element Interpolator	34

0.1 Forewords

This is an english translation of the course of the University of Paris 6, entitled "CALCUL SCIENTIFIQUE EN C++" and given by *Dominique Bernardi* and *Olivier Pironneau*. It is on the web, in French since February 1997. It is part of a new experience for teaching C++ and Numerical Analysis. This English translation is incomplete. The following books cover more or less the content of the course:

- - introduction to Scientific Computing for engineers by B. Lucquin and O. Pironneau (Wiley 98)

- - G. Buzzi-Ferrari: Scientific C++ (Addison-Wesley 1993) or
- - B. Stroutrupp: The C++ programming language.

REMARK 1

This html file has been prepared with latex2html, the LateX or postscript file can be sent on request by email to pironnea@asci.fr. The French html files are in the infosci

The course consists in one lesson per week and a compulsory exercise (a C++ program in general) which must be sent to the professor for check and correction.

0.2 Dates

0.3 Why go on the web?

The web is a good complement to books; in addition one can

- dialog with the professor
- correct the exercises
- use hypertext to browse the notes

0.4 Diploma

The professors (Bernardi-Pironneau) are willing to give a written letter explaining the effort given by the student.

0.5 Who correct the exercises?

D.Bernardi, O. Pironneau and C. Prud'homme.

0.6 The content

This one semester course is on

- The C++ language (*)
- An introduction to the theory of computer languages
- An introduction to computer graphics
- Basic UNIX commands (*)

- Some simple C functions from the Xwindows library (*)
- **Programming the Finite Element Method.**
- Compact storage, direct and iterative methods to solve $Ax=b$.
- The linear Partial Differential Equations of Physics
- The Finite Difference Methods for time dependent problems.
- The GMRES method for nonsymmetric or nonlinear problems

You are expected to spend about 12 hours on each lesson including exercises and programs. Items marked with a (*) have no written notes and are learned "de facto" by practice while doing the rest.

The final stage is a project, similar to freefem though shorter. This year an introduction to automatic differentiation within freefem will be the project.

0.7 What you need to do.

- 1. Check that you have access to a unix machine either on the web or, if you are a PC or Mac owners you should look seriously into the free unix system for PCs: linux (see also other sites.) It is possible to work on Windows 95 with Visual C++ or on a Mac with MetroWerks CodeWarrior C++ compilers but some of the exercises will not be possible with these.
- 2. Check you have a C++ compiler, preferably gcc 2.7 g++ (it is free) under X. It is also a good idea to have a debugger, for instance (gdb 4.16 with gcc), and of course a text editor such as (vi under unix or better xemacs)
- 3. Install and check on an example freefem
- 4. Install and check on an example gnuplot 3.5
- 5. Check you have access to the 2 books mentioned above.

1 LESSON: Introduction

1.1 Orientation

We begin by an exercise to learn how to read/write a file and then proceed with the numerical solution of a one dimensional time dependant heat

equation by the finite difference method. We will use 3 numerical schemes: the explicit Euler scheme the implicit Euler with the relaxation method and with the Gauss factorization method. All along errors and results can be plot with the software "gnuplot".

1.2 How to read a file

Assume that we have nv values of a function f stored in a file "f.dat" in the format

```
nv
f[0]
f[1]
...
f[nv-1]
```

Suppose we wish to display on the console the sum of the values of f . For this we would write in C++

```
// file sum.cpp
#include <iostream.h>
#include <fstream.h>

void main()
{
    ifstream ffile("f.dat");
    int nv;
    float sum=0;
    ffile >> nv;
    for(int i=0; i<nv; i++)
    { ffile >> fi;
      sum += fi;
    }
    cout << "sum=" << sum << endl;
}
```

On Unix machines, to compile the program with a compiler called `g++` one would type " `g++ sum.cpp` " and to execute the program one would type "a.out".

Should you need to store the values of f in an array, then you would use:

```
// file sum1.cpp
```

```

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

void main()
{
    ifstream ffile("f.dat");
    int nv;
    float sum=0;
    float* f;
    ffile >> nv;
    f = new float[nv]; // dynamic allocation of an array of floats
    for(int i=0; i<nv; i++)
    { ffile >> f[i];
      sum += f[i];
    }
    cout << "sum=" << sum << endl;
    delete [] f; // optional (automatic on exit)
}

```

Consider now the problem described in the introduction of the book [1]:
Find u solution of

$$\partial_t u - \kappa \partial_{xx} u + \alpha u = 0,$$

which is the equation for the temperature in a rod with air around at temperature 0 degree. Assume that initially the rod is at temperature $u_0=20$ and that the temperature of the left end is $u_1=20$ while at the right hand it is $u_2=100$.

Consider the Euler explicit finite difference scheme for

$$u_j^n \approx u(j\delta x, n\delta t) : \frac{1}{\delta t}(u_j^{n+1} - u_j^n) = \frac{\kappa}{\delta x^2}(u_{j+1}^n - 2u_j^n - u_{j-1}^n) - \alpha u_j^n = 0$$

If we decide to store u_j^{n+1}, u_j^n in the same memory, this is trivially implemented as:

```

#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include <iostream.h>

const int M = 100;

```

```

class data
{
    public:
    float kappa, u0, u1, u2, alpha;
    void read();
};

void data::read()
{
    cout << "enter kappa:"; cin >>kappa;
    cout << "enter u0:"; cin >> u0;
    cout << "enter u1:"; cin >> u1;
    cout << "enter u2:"; cin >> u2;
    cout << "enter alpha:"; cin >> alpha;
}

class solution
{ public:
float * u;
solution(const int M);
void init(data& d);
void explicit( data& d);
void result();
};

solution::solution(const int M)
{
u = new float[M+1];
}

void solution::init( data& d)
{
for(int i = 0; i< M; i++)
u[i]=
}

void solution::explicit(data& d)
{
    const float kappadx2 = dt*d.kappa / dx / dx;
    for(int i = 1; i< M-1; i++)

```

```

    u[i] += kappadx2*(u[i+1] - 2*u[i] + u[i-1]) -dt*alpha*u[i];
}

void solution::result()
{

}

void main()
{
solution u(M);
data d;

d.read();
u.init(d);

for (int n = 0; n < nmax; n++)
    u.explicit(d);

u.result();
cout<<"fin";

}

```

EXERCISE 1 (*Compulsory*)

Fill out the gap and execute the program for 3 values of M and

$$\kappa = \alpha = 0.1, \quad \delta t = 0.8\delta x^2/(2\kappa).$$

Remember that the scheme may be unstable if $\delta t > \delta x^2/(2\kappa)$.

Plot the log of the error versus the log of M for 3 values of M. For this you will need the analytical solution of the problem and how to plot a function with gnuplot; gnuplot is explained below. Here is one analytical solution:

$$u(x, t) = e^{-\alpha t}(u_2x + u_1(1 - x)) + e^{-(\alpha-\beta)t} \sinh\left(x\sqrt{\frac{\alpha - \beta}{\kappa}}\right)$$

with

$$u^0 = u_2x + u_1(1 - x) + \sinh\left(x\sqrt{\frac{\alpha - \beta}{\kappa}}\right).$$

It works for any $\beta \leq \alpha$.

1.3 Gnuplot

To visualize the solution you must create a file containing on each line the values of x and y of the function $y(x)$ you want to plot. Then if the file containing these data is called "ex.gnu", at the unix level you can type

```
gnuplot
plot "f.gnu" w l
quit
```

and it will display a window with a graph of the function.

1.4 batch file

One may want to chain the task of execution of the program and display the function. For this there are two solutions; one is to add into the C program a line with

```
exec("gnuplot | plot 'f.gnu' w l")
```

If it does not work try replace `exec` by "system".

The second method is to create a batch file, i.e. a text file say "mybatch" which contains the lines

```
a.out
gnuplot
plot "f.gnu" w l
```

and to change the access rights of the file "mybatch" by

```
chmod 777 mybatch
```

to allow the command

```
mybatch
```

to work.

2 LESSON on freefem and gnuplot

2.1 Orientation

The software freefem has been conceived to ease the preparation and visualisation of data for 2D problems such as partial differential equation solvers.

But freefem is also a good example of the kind of software that can be made once the techniques explained in this course are mastered. Let us

begin by playing with the language Gfem which is at the heart of freefem. Then we shall show how the rudiment graphics of freefem can be expanded with another graphic package: gnuplot .

2.2 Problem statement

Suppose we wish to visualize

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

Recall that a function is not just a formula which yields values for given values of x,y; it is also the domain of definition, here

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

To represent the domain we will use an approximation, which is the union of triangles numbered over a set of integers I.

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

A triangulation is such a covering with the following properties (cf figure 1):

- The union of all triangles is an approximation of the domain
- Two distinct elements (triangles) can intersect only as a FULL edge or a vertex.
- Corners and other singular points of the continuous domain must be vertices of the approximate domain.
- Vertices of the approximate boundary must be on the continuous boundary.

Note that the continuous domain and the approximate domain are equal if and only if the first one is polygonal.

2.3 An example with freefem

We shall define f and the triangulation of its domain of definition by writing a small program which will be stored in a file called "trace.pde":

```
/* file 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);
```

This text follows the Gfem syntax. It defines a border with reference number 1 by a parametric representation where the parameter t goes from 0 to 2π and has $n=50$ vertices on it.

The mesh generator implements a Delaunay-Voronoi algorithm and has a maximum vertex number security features so as to avoid over flow. Here it will stop if it ever generates more than a 1000 vertices.'

Then the triangulation is stored in a file called "trace.msh" for future use. The format will be described below.

The function f is then defined and its values at the vertices are stored in a file called "trace.dta". Its format

- number of values (vertices)
- value of f on each vertex.

REMARK 2

Beware that older version of freefem may have two values of f on each line; an old reminder that f could be complex valued and so the second value was the complex component.

EXERCISE 2 *Execute the previous program for several values of n . Under unix you should type*
freefem trace.pde

Figure 1: Results

2.4 Triangulation file

A triangulation is defined by

- The 2 coordinates of each vertex.
- The 3 vertex numbers of each triangle.

As vertices are stored in an array (a line for each vertex in file trace.msh), they have a number: the line position in the file, for instance.

Similarly, the very fact of specifying each triangle by a line in the file "trace.msh", means that triangles have been numbered.

In addition, it is convenient to associate to each vertex and to each triangle a *reference number*. For vertices it indicates on which boundary it is, with the convention that 0 indicates an interior point.

To read a triangulation file we must first define a standard format:

- Number of vertices, number of triangles.
- One line per vertex with the 2 coordinates x, y et an integer "where".
- One line per triangle with the numbers for its 3 vertices and an integer "where".

Each vertex receives a number corresponding to its line position in the file; this number can be used to identify the vertex. The number "where" is a convenience to find out quickly if a vertex is strictly inside the domain (where=0) or on the boundary. The number "where" for the triangle is convenient when there are several medium in the domain, such as on iron part and one copper part in electromagnetism for instance.

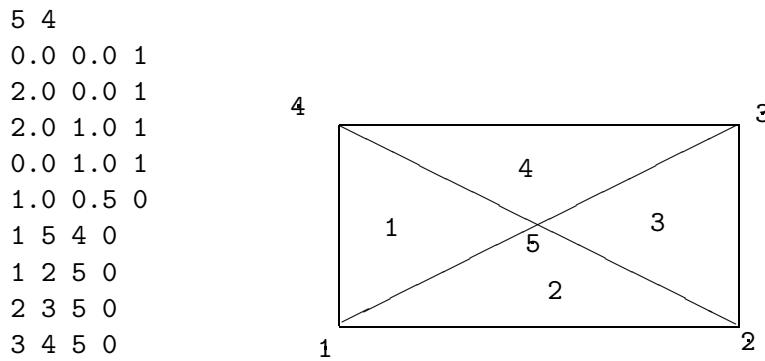


Figure 2: A tiny triangulation and its file of data.

2.5 Read a Triangulation file

In C++ we use the "class" structure to define a vertex and a triangle

```

// file readT.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");
}

```

REMARK 3

In FORTRAN all arrays begin at 1 (unless specified otherwise) but in C they begin at 0. When a file format is defined it is better to adjust to the FORTRAN convention because C programmers know about it but others don't. So a one is subtracted from $t[j].v[k]$ in the instruction "- i1".

2.6 Display a Triangulation with gnuplot

We do not describe the public domain program gnuplot in details but show how to use it in this context.

For instance, if the file "ex.gnu" contains

```

0.0 0.0
0.0 0.1
0.1 0.05

0.1 0.05
0.0 0.0

```

then the command

```

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

```

will display in a separate window a rectangle. of length 0.1 and size 0.05. The plot is performed by drawing first a polygonal line made by the first 3 sides and then the last side is drawn as a segment.

More precisely, a blank line creates a "moveto" to the point of the next line; otherwise the next line is a "lineto".

The directive `[-0.2,0.2]x[-0.2,0.2]` indicates that we wish to display only whatever falls in this square. This is useful to make "zoom".

Obviously our triangulation files and our gnuplot files are different. Let us write a translator from the first to the second. This is very helpful to display a file created by `freem`

```
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;
    }
    f.close();
}
```

REMARK 4

For a function the type "const" (at the end of its declaration) means that the function does not modify the fields of the class.

We mustn't forget also to add to the class definition the line

```
void Triangulation::gnu(const char* filename) const
```

Therefore the main function will be

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

REMARK 5

There is another public domain plotting program "plotmtv" which also reads gnuplot formats.

EXERCISE 3

Write a program which reads a triangulation file in the format freefem and generate a file in the gnuplot format.

EXERCISE 4

- 1. Write a Gfem program which generates the triangulation of a square with an elliptic whole inside (wholes are obtained by clockwise description of its boundary instead of counterclockwise).
- 2. Make the direction counterclockwise in the ellipse, give it number 0 and see what buildmesh does. Note that it can be used to control the triangulation of a square and refine one part.
- 3. In this way refine the lower part of a square.

2.7 Vizualisation of a function

Gnuplot is capable of 3D plot of straight lines. But then each segment is described by two points with 3 coordinates each. To break the plot of a polygonal line (i.e. insert a moveto) we need here too an empty blank line, but it seems that some machine need 2. So we get the following example which plots $f(x, y) = x^2 + y^2$:

```
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++)
  {
```

```

        for(int i=0; i<3; i++)
            f << g.v[t[i].v[i]].x <<"\t"<< g.v[t[i].v[i]].y <<"\t"
                << u[t[i].v[i]] << endl;
        f << g.v[t[i].v[0]].x << "\t" << g.v[t[i].v[0]].y <<"\t"
            << u[t[i].v[0]] <<endl<<endl<<endl;
    }
}

```

Then gnuplot must be launched as follows:

```

gnuplot
set parametric
splot "fmicro.gnu" w l

```

2.8 Complement: the debugger gdb

Under Unix the gnu debugger is called gdb or dbx or other; ther is a advance version called ddd. The basic operation of these debugger is described here.

- Compile the program with option "-g". then launch gdb with the program exec file as parameter. Exemple

```

g++ -g readT.cpp
gbd a.out

```

- Try the commandt "help". See your source file by typing "list" or "l".
- Put a break point at instruction 49 (Triangulation g("micro.msh");).Then do a step by step execution by typing "s".
- Try to see the value of a variable by type "p " and its name.

This gives the following sequence

```

g++ -g lirmicro.cpp
gbd a.out
l
b 49

```

```
run
s
s
s
p nv
...
quit
```

2.8.1 The command where

If the program fails, the debugger will tell you where. But to know the line where it fails is not always enough because it can be within a function which is called at many different places. We must also know the sequence of function call. The easiest for that is to use the standard C function "atexit" from the library stdlib:

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

A breakpoint should be put on the line "cout<<"Fin"..." so that the debugger will stop on this line if there is an interruption caused by the Unix system. The effect of the function atexit(func) is to force the execution of the function whose name is the parameter in case of system interrupt or normal exit. After that, typing a "where" will give the history of function calls.

2.8.2 Parameter passing in UNIX

we now have a tool, let us call it fem2gnu to translate a .msh file into a .gnu file. It would be nice to be able to type

fem2gnu micro

to translate the file "micro.msh" into a "micro.gnu" file.
The following program solves the problem

```
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 Display of the level lines of a function

2.9.1 Principle: P1 interpolation

If the domain is triangulated (divided into triangles which cover the domain, and do not intersect in other thing than a vertex or a whole side) the easiest is to approximate the function f by its "P1 interpolate". It is the unique function which

- is continuous,
- is linear within each triangle
- is equal to f at the vertices.

Theorem 1 *The P1 interpolate of f is uniquely defined by the values of f at the vertices.*

Proof

If a point x is within a triangle T with vertices q^i then it can be represented by its 3 "barycentric coordinates":

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

This is 3 equations for 3 unknowns λ_i , the determinant of this linear system is twice the area of the triangle so it is non-zero and the λ_i are unique. Then the interpolate is

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

If x is on an edge, the interpolate of one of the barycentric coordinates is zero so it is

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

The "level lines" of f are

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

When f is replaced by its interpolate, the level lines are polygonal and on each triangle it is a line segment.

An intersection point q between a level line and an edge of the triangulation will be found if for some i, j, a we have:

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

So the principle for the drawing of a portion of level line is as follows.

```
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);
    }
}
}

```

It is necessary to check that the denominators are not zero and the drawing of the border of the domain must be added.

EXERCISE 5 (*Compulsory*)

Write a program which reads a triangulation in the freefem format. Write a program which reads a file of values of f at the vertices and then generates a gnuplot file for the display of level lines. The presence of 2 parameters in the unix command must be tested. If there is only one parameter; if only one parameter is there, then the program will decide that it is a mesh plotting request and do it.

2.9.2 Error estimates

In [?] (Theorem 6.1 p71) it is shown that the H^1 error between the interpolate and the function is $O(h)$ while the L^2 error is $O(h^2)$.

EXERCISE 6

Take 3 triangulations by doubling each time the number of points on the boundary. Plot a function in the 3 cases, compute the L^2 errors and plot them versus h . Check that they are aligned on a line of slope 1 in a log-log scale.

3 Lesson: An explicit scheme for the wave eq.

3.1 The problem

The wave equation in the unit square with homogeneous Dirichlet data

$$\partial_{tt}u - \Delta u = 0, \quad \forall x \in \Omega = (0, 1) \times (0, 1), \quad \forall t \in (0, T), \quad u|_{\partial\Omega} = 0$$

has an exact solution

$$u = \sin(\pi x) \sin(\pi y) \sin(t\pi\sqrt{2})$$

It corresponds to the initial data

$$u|_{t=0} = 0, \quad \partial_t u|_{t=0} = \sin(\pi x) \sin(\pi y) \pi \sqrt{2}$$

3.2 The scheme

Then Crank-Nicolson scheme for the time derivative is

$$\frac{1}{\delta t^2}(u^{n+1} - 2u^n + u^{n-1}) - \Delta u^n = 0.$$

In variational form it is

$$\int_{\Omega} (u^{n+1} - 2u^n + u^{n-1})w = -\delta t^2 \int_{\Omega} \nabla u^n \nabla w, \quad \forall w \in H_0^1(\Omega); \quad u^{n+1} \in H_0^1(\Omega)$$

Discretized by the Finite Element method of degree 1 it is

$$\int_{\Omega} u^{n+1}w = \int_{\Omega} (2u^n - u^{n-1})w - \delta t^2 \int_{\Omega} \nabla u^n \nabla w, \quad \forall w \in V_{0h}; \quad u^{n+1} \in V_{0h}$$

with

$$V_h = \{w \in C^0(\overline{\Omega_h}) : w|_T \in P^1\}, \quad V_{0h} = \{w \in V_h : w|_{\partial\Omega_h} = 0\}$$

where T is any triangle of a triangulation of Ω and where Ω_h is the union of these triangles.

The solution is decomposed on the basis of hat function:

$$u^n(x) = \sum_i u_i^n w^i(x), \quad w^i \in C^0(\overline{\Omega_h}) : w^i|_T \in P^1, \quad w^i(q^j) = \delta_{ij}$$

where q^j denotes the vertices of the triangulation. We obtain

$$BU^{n+1} = B(2U^n - U^{n-1}) - \delta t^2 AU^n, \quad \text{with } B_{ij} = \int_{\Omega} w^i w^j, \quad A_{ij} = \int_{\Omega} \nabla w^i \nabla w^j$$

Finally, using the mass lumping approximation for the first integral

$$B_{ij} \cong \sum_k \frac{|T_k|}{3} \sum_{l=0,1,2} w^i(q^{il}) w^j(q^{il}) = \frac{\delta_{ij}}{3} \sigma_i, \quad A_{ij} = - \sum_{k:q^i, q^j \in T_k} |T_k| h_{ik}^{\vec{}} \cdot h_{jk}^{\vec{}}$$

where σ_i is the sum of the areas of the triangles which have q_i as vertex and where h_{ik} is a vector in the direction orthogonal to the base of T_k opposite to q^i and of length the distance between this vertex and this base.

Concretely, for a triangle T_k with vertices q^i, q^{i+}, q^{i++} this vector is $\vec{h}_{ik} = (q_2^{i++} - q_2^{i+}, -(q_1^{i++} - q_1^{i+}))^T / (2|T_k|)$ Hence the scheme is written as

$$\sigma_j = \sum_{k:q^j \in T_k} |T_k|, \quad a_{ij}^k = (q^{j++} - q^{j+}) \cdot (q^{i++} - q^{i+}) / (4|T_k|)$$

$$u_i^{n+1} = 2u_i^n - u_i^{n-1} - \frac{3\delta t^2}{\sigma_i} \sum_{k:q^i, q^j \in T_k} a_{ij}^k u_j^n.$$

3.3 The program

To compute this solution we shall first write a program for freefem to construct a mesh

```
border(1,0,1,10) begin x:=t; y:=0 end;
border(1,0,1,10) begin x:=1; y:=t end;
border(1,1,0,10) begin x:=t; y:=1 end;
border(1,1,0,10) begin x:=0; y:=t end;
buildmesh(2000);
savemesh("square.msh");
```

Then we shall write a program which reads the mesh and implement the scheme above. Below the outline of the program is given:

```
#include <math.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <assert.h>
#include "aarray.h"
#include "grid.h"

const int itermax = 100;
void main()
{
float dt = 0.75/itermax, err = 0, pi = 4*atan(1.0);
Grid g("square.msh");

A<float> dt2sigma(g.nv), u2(g.nv), u1(g.nv), u0(g.nv);
```

```

for(int i=0; i<g.nv;i++) // initialization
{
    dt2sigma[i] = 0;
    u0[i] = 0;
    u1[i] = dt*pi*sin(g.v[i].x*pi)*sin(g.v[i].y*pi)*sqrt(2);
}
for(int k=0; k<g.nt;k++)
{
    g.t[k].area = ???;
    for(int iloc = 0; iloc<3; iloc++)
        dt2sigma[g(k,iloc)] += g.t[k].area/dt/dt/3;

for(int iter=0; iter < itermax; iter++) // time loop
{
for(int i=0; i<g.nv;i++)
u2[i] = 2*u1[i] - u0[i];

for(int k=0; k<g.nt;k++)
for(int iloc = 0; iloc<3; iloc++)
    if(!g.t[k].v[iloc]->where)
    {
        int i = g(k,iloc);
        int ip = g(k,(iloc+1)%3);
        int ipp= g(k,(iloc+2)%3);
        for(int jloc = 0; jloc < 3; jloc++)
        {
            int j = g(k,jloc);
            float aijk = ???;
            u2[i] -= aijk * u1[j]/dt2sigma[i];
        }
    }
for(int i=0; i<g.nv;i++)
{
    u0[i] = u1[i];
    u1[i] = u2[i];
}
}
for(int i=0; i<g.nv;i++)
    err += pow(u1[i] - sin(itermax*dt*pi*sqrt(2))
                *sin(g.v[i].x*pi)*sin(g.v[i].y*pi),2);
cout << sqrt(err)/g.nv << endl;
}

```

3.4 Exercise

EXERCISE 7 *Compulsory*

Complete the program above; run it for 3 triangulations and plot the log-log error curve. Plot also the result by using one of the method described in the previous lesson.

4 Lesson: An implicit scheme

4.1 The Crank-Nicolson implicit scheme

Consider again the wave equation of the previous section with zero right hand side and zero Dirichlet condition on the boundary. Explicit schemes as the one above usually have a stability condition. It can be more efficient at times to use an unconditionally stable scheme such as

$$\frac{1}{\delta t^2}(u^{n+1} - 2u^n + u^{n-1}) - \Delta \frac{u^{n+1} + u^{n-1}}{2} = 0.$$

EXERCISE 8

Work out the Von Neumann stability condition for this scheme in one dimension of space with a uniform finite difference discretization. Find also the error of consistency.

4.2 Discretization in space

In variational form it is: find $u^{n+1} \in H_0^1(\Omega)$ with

$$\int_{\Omega} (u^{n+1} - 2u^n + u^{n-1})w + \delta t^2 \int_{\Omega} \nabla \frac{u^{n+1} + u^{n-1}}{2} \nabla w = 0, \quad \forall w \in H_0^1(\Omega).$$

Discretized by FEM of degree 1 it is: find $u^{n+1} \in V_{0h}$ such that

$$\int_{\Omega} u^{n+1}w + \frac{\delta t^2}{2} \int_{\Omega} \nabla u^{n+1} \nabla w = \int_{\Omega} (2u^n - u^{n-1})w - \frac{\delta t^2}{2} \int_{\Omega} \nabla u^{n-1} \nabla w, \quad \forall w \in V_{0h},$$

with

$$V_h = \{w \in C^0(\overline{\Omega_h}) : w|_T \in P^1\}, \quad V_{0h} = \{w \in V_h : w|_{\partial\Omega_h} = 0\}$$

where T is any triangle of a triangulation of Ω and where Ω_h is the union of these triangles.

The solution is written on the basis of hat functions:

$$u^n(x) = \sum_i u_i^n w^i(x), \quad w^i \in C^0(\overline{\Omega_h}) : w^i|_T \in P^1, \quad w^i(q^j) = \delta_{ij}$$

where $\{q^j\}_j$ denotes the vertices of the triangulation and the summation extends to INNER vertices only, because u is zero on the boundary. We obtain

$$AU^{n+1} = 2BU^n - AU^{n-1}, \quad B_{ij} = \int_{\Omega} w^i w^j, \quad A_{ij} = \int_{\Omega} w^i w^j + \frac{\delta t^2}{2} \int_{\Omega} \nabla w^i \nabla w^j$$

where the U are vectors of values of u on vertices for all i, j corresponding to inner vertices.

The template for this algorithm is

```

initialize or read data: grid g, u0, u1, dt...
initialize U0 and U1
loop on n
{
    Compute BU1
    Compute AU0
    Solve AU = 2(BU1) - AU0
    do U0=U1, U1=U
}
Write U on file for graphic display.
```

The initialization step is done as before:

```

#include <math.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <assert.h>
#include "aarray.h"
#include "grid.h"

const int itermax = 100;
void main()
{
float dt = 0.1;
Grid g("square.msh");
```

```
A<float> u0(g.nv), u1(g.nv), u(g.nv);
...
```

Here to compute BU there is no need to use mass lumping, we can use an exact formula:

$$(BU)_j = \int u_h w^j = \sum_k \frac{|T_k|}{3} \sum_{i=0,1,2} \frac{u^i + u^{i+}}{2} \frac{w^j(q^i) + w^j(q^{i+})}{2}$$

which is also

$$BU_j = \sum_k (u^j \frac{|T_k|}{12} + \sum_{i=0,1,2} u^i \frac{|T_k|}{12})$$

It is programmed by "assembling" the contributions triangle by triangle. Note that it is easier to extend the array u to boundary vertices and put these extra values to zero rather than test the field g.v[i].where.

```
void bMul(A<float>& bu, Grid& g, A<float>& u)
{
    for(int i=0; i<g.nv; i++)
        bu[i] = 0;

    for(int k=0; k<g.nt;k++)
        for(int iloc = 0; iloc < 3; iloc++)
            { int i = g(k,iloc);
              for(int jloc = 0; jloc<3; jloc++)
                  bu[i] += u[g(k,jloc)] * g.t[k].area / 12;
              bu[i] += u[i] * g.t[k].area / 12;
            }
}
```

Similarly AU is computed as in the explicit scheme

```
void aMul(A<float>& au, Grid& g, A<float>& u, float dt22)
{
    for(int i=0; i<g.nv; i++)
        au[i] = 0;

    for(int k=0; k<g.nt;k++)
        for(int iloc = 0; iloc < 3; iloc++)
```

```

{ int i = g(k,iloc);
  int ip = g(k,(iloc+1)%3);
  int ipp= g(k,(iloc+2)%3);
for(int jloc = 0; jloc<3; jloc++)
{ int j = g(k,jloc);
  int jp = g(k,(jloc+1)%3);
  int jpp= g(k,(jloc+2)%3);
  float aijk = ((g.v[jpp].x - g.v[jp].x)*(g.v[ipp].x - g.v[ip].x)
                +(g.v[jpp].y - g.v[jp].y)*(g.v[ipp].y - g.v[ip].y))
                /g.t[k].area/4;
    au[i] += u[j] * (g.t[k].area / 12 + aijk * dt22 );
  }
  au[i] += u[i] * g.t[k].area / 12 ;
}
}

```

where dt22 is dt*dt/2 and where aijk is the same as in the explicit scheme:

$$aijk = \int_{T_k} \nabla w^i \nabla w^j$$

4.3 The conjugate gradient algorithm

To solve the linear system $AU=F$ we use the conjugate gradient algorithm:

```

initialization: U=U0 (ex U0=0), H = G = F- AU0, eps = 1e-6
iterations:    d = |G|^2
               r = G.H / H.AH
               U := U + r H
               G := G - rAH
               c = |G|^2/d
               H := G + c H
               if(|G|^2<eps |F|^2) stop

```

Here we notice that

$$(AH)_i = \int_{\Omega} w^i h + \frac{\delta t^2}{2} \int_{\Omega} \nabla w^i \nabla h$$

where $h = \sum h_i w^i$, so that this is programmed as

Scalar products and norms are programmed as usual

```

float scal = 0, norm2 = 0;
for(int i=0; i<g.nv; i++)
{ scal += a[i]* b[i]; // scalar product a * b
  norm2 += pow(a[i],2); // norm square of a
}

```

4.3.1 Remark for non zero Dirichlet conditions

If U is Ug on the boundary instead of being zero, the theory asks us to work with $U' = U - Ug$ and use the previous method on $AU' = F' = F - AUg$.

But we notice that an instruction like $U' := U' + r H$ is identical to $U := U + r H$ if H is zero on the boundary nodes where Ug is non zero. Furthermore $U' = 0$, $H = G = F'$ is identical to $U = Ug$, $H = G = F - A U$. Consequently we can work with U directly provided U has the right values at the Dirichlet boundaries and provided that H and G are always 0 on these boundaries.

5 Lesson: Automatic triangulation

5.1 Automatic triangulation of a square

5.1.1 The problem

Assume that we are given $N+4$ points the last four of which defines a square containing all the other points. The points are numbered $(q^0, \dots, q^{N-1}, q^N, q^{N+1}, q^{N+2}, q^{N+3})$

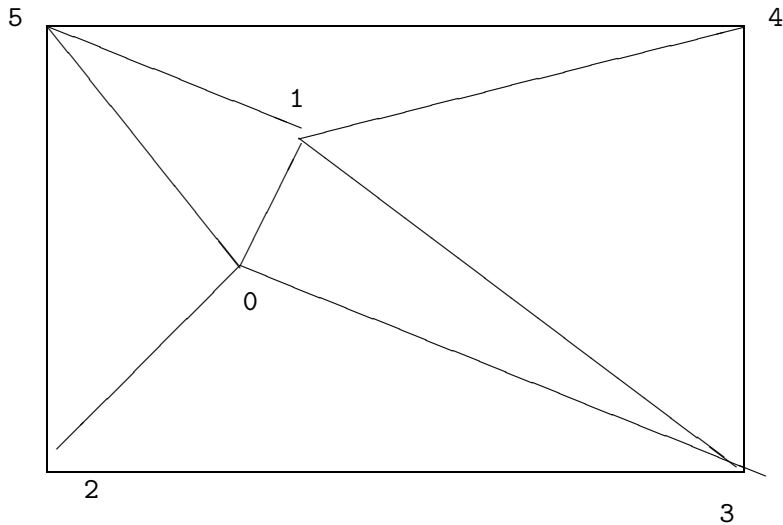
5.1.2 Algorithm

We maintain a list of triangles. At start it is the two triangles $(q^N, q^{N+1}, q^{N+2}), (q^{N+2}, q^{N+3}, q^N)$.

Then we perform a loop on the points, backward for simplicity:

for $i=N-1$ down to 0 do

- case 1: there exists a triangle of the list which contains q^i strictly. then replace this triangle by the 3 subtriangles which have q^i for vertex.
- case 2: the point q^i is on the border of the square. Then find the unique triangle which contains q^i and replace it by the 2 subtriangles which have q^i for vertex.
- case 3: There are two triangles which contain q^i (i.e. q^i is on an inner edge. Then each triangles must be replaced by the two sutriangles which have q^i for vertex..



5.1.3 The Delaunay criteria

The previous triangulation is admissible for the finite element method but it is a bad one, it "looks terrible". By this we mean that there are many obtuse angles and small triangles near to large ones.

Notice that to each inner edge of a triangulation we can associate a quadrangle made of the two triangles adjacent to the edge.

A triangulation is said to be "Delaunay" if for each edge the circle circumscribing one triangle does not contain the fourth vertex.

Edge swap If the 4 vertices of a quadrangle associated to an inner edge are not cocyclic then the of the two configurations obtained by swapping diagonals in the quadrangle, one is Delaunay, the other is not.

Proposition When a configuration becomes Delaunay by an edge swap the minimum angle in the 2 triangles increase.

Algorithm

Loop until nothing changes

Loop on the inner edges E

- Find the 2 triangles T_k, T_l adjacents to E denote q^1, q^2, q^3 the vertices of T_k and by q^2, q^1, q^4 those of T_l .
- Check the Delaunay criteria. If it fails replace T_k, T_l by the triangles q^3, q^4, q^1 and q^4, q^3, q^2

Proposition The algorithm converges.

Indeed at each loop the smallest angle increase. When it no longer increases then the next to smallest angle increase.... The number of configurations being finite the process converges.

5.2 Triangulation of a convex domain

Let D be a domain given as the convex hull of a set of points q^1, q^2, \dots, q^N

5.2.1 Algorithm

1. Choose 4 points q^{N+1}, \dots, q^{N+4} vertices of a rectangle and very far from $q^1 \dots q^N$.
2. Apply previous algorithm to $q^1 \dots q^{N+4}$
3. Remove all the triangles which have a vertex in $V_\infty = \{q^{N+1} \dots q^{N+4}\}$

If the q^{N+1}, \dots, q^{N+4} are far away enough then this algorithm works. Indeed, first note that there are no triangle with 3 vertices in V_∞ .

Then if a triangle has 2 vertices in V_∞ then the third one is necessarily on the border of D and so the triangle is entirely outside D and it can be removed.

Now if a triangle only one vertex in V_∞ tmust show that the border of D is made of edges of the triangulation.

Suppose a border edge is missing. Then consider the quadrangle made by the two end point of that edge, the vertex at infinity and the fourth vertex of the two triangles. This situation would violate the Delaunay criteria because the circle passing through the 3 vertices at finite distance leaves the fourth one outside as it is far away

5.3 Triangulation of a non convex domain

A non convex domain is defined by a set of oriented edges which defines its boundary. By hypothesis the domain is on the left of its oriented boundary.

We assume that the vertices of the boundary are in $V = \{q^1 \dots q^N\}$ and we seek a triangulation with vertices V .

5.3.1 Algorithm

1. Triangulate the convex hull of V with the previous algorithm
2. Edge forcing: If there is an boundary edge which is not an edge of the triangulation then consider all the triangles which intersect that edge and call P_l and P_r the polygons hence formed on the left and right of that edge.
3. Remove all the triangles in P and apply this algorithm recursively to P_l and P_r .
4. Loop untill all boundary edges are in the triangulation
5. Remove all triangles which are outside the domain..

To build the polygons P one way is to start from one vertex of the edge to be forced in, find the triangle which intersect it then take its neighbor by the opposite edge and so on. The data structure needed for this is

- For each vertex the list of all triangles containing the vertex.
- For each edge the left and right triangle which have the edge for side.

5.3.2 Generating interior points

In practice we use the previous algorithm without interior points; all vertices are on the boundary. We assume that the user has input his request on vertex density through the density of points on the boundary.

So each vertex has a weight. For boundary vertices it is the average length of the two surrounding boundary edges.

Then we perform the following test on each edge of the triangulation:

Fi the length of an edge is larger than the average weigth of its vertices then we dive the edge by adding a middle point and assigning to it the average of the weights of the two vertices of the edge.

Then the triangulation algorithm is applied again to the new set of points. And so on till no edge is divided.

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 specifiques (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);
}

```

The files mshptg.c et mshptg.h are accessible by ftp, together with their fortran version mshptg.f which was used to generate them (f2c). The author is (Frederic.Hecht@inria.fr).

EXERCISE 9 (*Compulsory*)

Test the program; add a small square hole inside.

Note that a makefile is necessary because mshptg.c needs to be linked with the math library (lm).

At execution, if the program prints a positive value for nt (the number of triangles) then it is a sign that it has worked. To make sure the file containing the new triangulation ctriangle.dta, must be checked.

5.5 Mesh Adaption

It is possible to adapt the mesh to a function f by redefining the distance between two points with the Hessian matrix $\nabla\nabla f$.

Let M be a positive definite $d \times d$ matrix ($d=2$ in 2D).

The distance associated with M is

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

The Delaunay criteria will now be applied with this new distance. Notice that circles become ellipses. In the algorithms above the notion of distance appears also when the edges are divided.

We recall that the interpolation error from u_h to u is bounded by

$$\|u - u_h\|_{E,0} \leq c \|\nabla\nabla u\|_{E,\infty} h^2$$

where E recalls that the Sobolev norms are defined with the Euclidian distance. for L^2 and for L_∞ .

Thus we see that the error is large if $\|\nabla\nabla u\|$ is large. So the idea is to keep $\|h^T \nabla\nabla u h\|$ small. Note however that there are no error estimates which take into account the anisotropy of $\nabla\nabla u$. So the proof of the efficiency of this approach is open.

Similarly if we wish to adapt the mesh to two function we can consider the matrix $(\nabla\nabla u)(\nabla\nabla v)$.

However $x^T M y$ is a proper scalar product only if M is positive definite.

Given any symmetric matrix we can compute its eigen values and eigen vectors and construct a positive definite matrix from it by

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

where Q is the rotation matrix which makes M diagonal and ϵ is a small number.

In practice the function u which is used to adapt the mesh is computed on a

coarse mesh. So there are several interpolations to do, the moment we pass from one mesh to another. This is a difficult task.

REMARK 6 N Metric Specified

The user could also wish to specify a function but not a metric. Then we can still use the previous approach but with the identity matrix multiplied by the user-specified function $h(x, y)$. This will generate an isotropic refinement with density adjusted to h .

5.6 A Fast Finite Element Interpolator

In practice one may discretize the variational equations by the Finite Element method. Then there will be one mesh for Ω_1 and another one for Ω_2 . The computation of integrals of products of functions defined on different meshes is difficult. Quadrature formulae and interpolations from one mesh to another at quadrature points are needed. We present below the interpolation operator which we have used and which is new, to the best of our knowledge. Let $\mathcal{T}_h^0 = \cup_k T_k^0, \mathcal{T}_h^1 = \cup_k T_k^1$ be two triangulations of a domain

Ω . Let

$$V(\mathcal{T}_h^i) = \{C^0(\Omega_h^i) : f|_{T_k^i} \in P^1\}, \quad i = 0, 1$$

be the spaces of continuous piecewise affine functions on each triangulation.

Let $f \in V(\mathcal{T}_h^0)$. The problem is to find $g \in V(\mathcal{T}_h^1)$ such that

$$g(q) = f(q) \quad \forall q \text{ vertex of } \mathcal{T}_h^1$$

Although this is a seemingly simple problem, it is difficult to find an efficient algorithm in practice. We propose an algorithm which is of complexity $N^1 \log N^0$, where N^i is the number of vertices of \mathcal{T}_h^i , and which is very fast for most practical 2D applications. **Algorithm**

The method has 5 steps. First a quadtree is built containing all the vertices of mesh \mathcal{T}_h^0 such that in each terminal cell there are at least one, and at most 4, vertices of \mathcal{T}_h^0 .

For each q^1 , vertex of \mathcal{T}_h^1 do:

- **Step 1** Find the terminal cell of the quadtree containing q^1 .
- **Step 2** Find the the nearest vertex q_j^0 to q^1 in that cell.

- **Step 3** Choose one triangle $T_k^0 \in \mathcal{T}_h^0$ which has q_j^0 for vertex.
- **Step 4** Compute the barycentric coordinates $\{\lambda_j\}_{j=1,2,3}$ of q^1 in T_k^0 .
 - – if all barycentric coordinates are positive, go to Step 5
 - – else if one barycentric coordinate λ_i is negative replace T_k^0 by the adjacent triangle opposite q_i^0 and go to Step 4.
 - – else two barycentric coordinates are negative so take one of the two randomly and replace T_k^0 by the adjacent triangle as above.
- **Step 5** Calculate $g(q^1)$ on T_k^0 by linear interpolation of f :

$$g(q^1) = \sum_{j=1,2,3} \lambda_j f(q_j^0)$$

- **End**

Two problems needs to solved:

- • *What if q^1 is not in Ω_h^0 ?* Then Step 5 will stop with a boundary triangle. So we add a step which test the distance of q^1 with the two adjacent boundary edges and select the nearest, and so on till the distance grows.
- • *What if Ω_h^0 is not convex and the marching process of Step 4 locks on a boundary?* By construction Delaunay-Voronoi mesh generators always triangulate the convex hull of the vertices of the domain. So we make sure that this information is not lost when $\mathcal{T}_h^0, \mathcal{T}_h^1$ are constructed and we keep the triangles which are outside the domain in a special list. Hence in step 5 we can use that list to step over holes if needed.

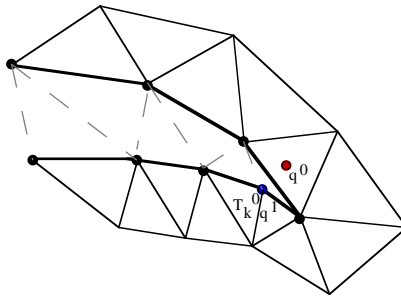


Figure 1. To interpolate a function at q^0 the knowledge of the triangle which contains q^0 is needed. The algorithm may start at $q^1 \in T_k^0$ and stall

on the boundary (thick line) because the line q^0q^1 is not inside Ω . But if the holes are triangulated too (dotted line) then the problem does not arise.

Remark Step 3 requires an array of pointers such that each vertex points to one triangle of the triangulation.

References

- [1] B. Lucquin and O. Pironneau : Introduction to Scientific Computing for engineers (Wiley 98)
- [2] G. Buzzi-Ferrari: Scientific C++ (Addison-Wesley 1993)
- [3] B. Stroutrupp: The C++ programming language III ed (Addison-Wesley 1997) .