# Numerical Methods For PDE : Assignement 5

Leo-Paul BAUDET

30 décembre 2023

## Table des matières

In that assignement we are interesting to solve the Laplace equation using mesh adaptivity. Let's first recall the problem we are interesting, with $\Omega = [0,1]^2$ :

$$\begin{cases} \nabla u = -f \text{ on } \Omega \\ u = u_d \text{ on } \partial\Omega \end{cases} \tag{1}$$

Here we know that the analytical solution is the following :

$$u(x,y) = g(x)g(y) \tag{2}$$

where

$$g(x) = (4x(1-x))^1 0 \tag{3}$$

The source term is the following

$$f = -g''(x)g(y) + g''(y)g(x) \tag{4}$$

with

$$g''(x) = 90 * (4-8x)^2(4x-4x^2)^8 \tag{5}$$

What we want to do is to implement an algorithm which refine the mesh until the error is smaller than a tolerance $\tau$. There are two ways to refine the mesh : do a uniform refinement and an adpatative refinement. The aim of this assignement is to compare the efficientcy of the two methods.

In order to do that we need to compute the error of each refined mesh. Here we know the analytic expresion of the solution, so we can easily compute the L2 or H1 norm. However, in most of the cases, we don't know the analityc expression of the solution, this is why we are going to use a ZZ-estimator of the H1-seminorm (which doesn't requires the expression of the analytical solution). In order to see that this estimator is well implemented, we are going to compare the error using the estimator and the real H1-sem norm error.

First of all, we are going to design two algorithms (using in a first time the exact H1-semi norm) : one which refine the mesh uniformaly until we reach an error greater than a tolerance $\tau$ and an other which use an adaptative refinement of the mesh.

Then in a second part, we are going to implement the computation if the H1-seminorm (using the analytical expresion of the solution), then implement the ZZ-estimator and compare the two errors.

# 1 H1 semi-norm

First of all, in order to designed the algorithms, we need to be able to compute the exact H1-semi norm (knowing the analytical solution). The H1-semi norm is defined as follow :

$$||u||_{H^1} = \left( \int |\nabla u|^2 \, dx \right)^{\frac{1}{2}} \tag{6}$$

Because we have computed the H1 norm in the first assignement, it's easy to compute the H1 semi norm (just removing the computation of $\int_\Omega |u|^2 d\Omega$). Here the code :

```
function [H1err]=compute_H1_semi_error(u, X, T, theReferenceElement)

IPweights = theReferenceElement.IPweights;
IPcoord = theReferenceElement.IPcoord;
N = theReferenceElement.N;
Nxi = theReferenceElement.Nxi;
Neta = theReferenceElement.Neta;

nOfElements = size(T, 1);
H1err = 0;

for i = 1:nOfElements
    Te = T(i, :);
    Xe = X(Te, :);
    xe = Xe(:, 1);
    ye = Xe(:, 2);
    Ue = u(Te);

    for g = 1:length(IPweights)
        N_g = N(g, :);
        Nxi_g = Nxi(g, :);
        Neta_g = Neta(g, :);

        J = [Nxi_g * xe, Nxi_g * ye; Neta_g * xe, Neta_g * ye];

        grad_ref = [Nxi_g; Neta_g];

        grad = J \ grad_ref;

        Nx_g = grad(1, :);
        Ny_g = grad(2, :);

        dvolu = IPweights(g) * det(J);
        Xg = N_g * Xe;


        errorH1 =(analytic_deriv_x(Xg) - Nx_g * Ue)^2 + (analytic_deriv_y(Xg) - Ny_g * Ue)^2;
        H1err = H1err + errorH1 * dvolu;
    end
end

    H1err = sqrt(H1err);
end
```

# 2 Refinement of mesh

## 2.1 Uniform refinement

Here what we want to do is to refine the mesh until the H1-semi-norm-error is smaller than a tolerance $\tau$. In order to refine the mesh, we use a function that is given to do that. We give to the function a list of elements that we want to refine. Here, we just have to give all the element of the mesh in order to refine it uniformaly. Here the algorithm :

```
function [u,mesh,error_semi,h]=unif_refinement(tau,theReferenceElement)
doPlot=1;
error_semi=[];
h=[];

[mesh]=generateMeshSquare(doPlot);
u=compute_sol(mesh.X,mesh.T,theReferenceElement);
error=compute_H1_semi_error(u,mesh.X,mesh.T,theReferenceElement);

error_semi=[error_semi,error];
num=size(mesh.X,1);
h=[h,1/num^2];
```

```
13
14 while error^2>tau
15     listElems=1:size(mesh.T,1);%numéro des éléments dans le maillage
16     mesh=refineListElements(mesh,listElems,doPlot);
17     u=compute_sol(mesh.X,mesh.T,theReferenceElement);
18     error=compute_H1_semi_error(u,mesh.X,mesh.T,theReferenceElement);
19
20     error_semi=[error_semi,error];
21     num=size(mesh.X,1);
22     h=[h,1/num^2];
23 end
24
25 end
```

First, line 6 we begin with an initial mesh, line 7 we compute the solution and line 8 we compute the error.

While the previous error is greater that the tolerance $\tau$ (line14), we refine all the elements of the mesh (line 15-16) and we recompute the solution and the error.

Moreover, we store the error in order to plot it for each step of the refinemeent. As it is indicated, a good indicator for the size of the refined mesh is $\frac{1}{N^{1/d}}$ where d is the dimension and N the number of nodes of the mesh.

### 2.1.1 Example

For example, we can take $\tau = 10^{-2}$ and see how many elements and nodes are necessary to get an error smaller than $\tau$. What we obtain is that we need 16641 nodes and 32768 elements.

## 2.2 Adaptative refinement

Here we want to refine the mesh in a more clever way. What we are going to do is to compute the H1-semi-norm error for each element, and if the current element get an error greater than $\tau$ then refine it. In order to do that, we need to modify a litle bit the computation of the previous H1-semi-norm

### 2.2.1 Elemental H1-semi-norm

Here we just have to remove the sum all over the elements, here the code :

```
1  function [H1err]=compute_H1_semi_error_elemental(u, Xe, Te, theReferenceElement)
2
3  IPweights = theReferenceElement.IPweights;
4  IPcoord = theReferenceElement.IPcoord;
5  N = theReferenceElement.N;
6  Nxi = theReferenceElement.Nxi;
7  Neta = theReferenceElement.Neta;
8
9  H1err = 0;
10 xe = Xe(:, 1);
11 ye = Xe(:, 2);
12 Ue = u(Te);
13
14 for g = 1:length(IPweights)
15     N_g = N(g, :);
16     Nxi_g = Nxi(g, :);
17     Neta_g = Neta(g, :);
18
19     J = [Nxi_g * xe, Nxi_g * ye; Neta_g * xe, Neta_g * ye];
20
```

```
21      grad_ref = [Nxi_g; Neta_g];
22
23      grad = J \ grad_ref;
24
25      Nx_g = grad(1, :);
26      Ny_g = grad(2, :);
27
28      dvolu = IPweights(g) * det(J);
29      Xg = N_g * Xe;
30
31
32      errorH1 =(analytic_deriv_x(Xg) - Nx_g * Ue)^2 + (analytic_deriv_y(Xg) - Ny_g * Ue)^2;
33      H1err = H1err + errorH1 * dvolu;
34  end
35      H1err = sqrt(H1err);
36  end
```

### 2.2.2 Algorithm

Then, we can apply the algorithm explained before, here the code :

```
1  function [u,mesh,error_semi,h]=adaptative_refinement(tau,theReferenceElement,analytic)
2  doPlot=1;
3  error_semi=[];
4  h=[];
5
6  [mesh]=generateMeshSquare(doPlot);
7  u=compute_sol(mesh.X,mesh.T,theReferenceElement);
8  error=compute_H1_semi_error(u,mesh.X,mesh.T,theReferenceElement);
9
10 error_semi=[error_semi,error];
11 h=[h,1/size(mesh.X,1)^2];
12
13 if analytic
14     while error^2>tau
15         listElems=[];
16         for i=1:size(mesh.T,1)
17             Te=mesh.T(i,:);
18             error_elem=compute_H1_semi_error_elemental(u,mesh.X(Te,:),Te,theReferenceElement);
19             if error_elem^2>10^-5
20                 listElems=[listElems,i];
21             end
22         end
23         mesh=refineListElements(mesh,listElems,doPlot);
24         u=compute_sol(mesh.X,mesh.T,theReferenceElement);
25         error=compute_H1_semi_error(u,mesh.X,mesh.T,theReferenceElement);
26
27         error_semi=[error_semi,error];
28         h=[h,1/size(mesh.X,1)^2];
```

Line 6, we generate an initial mesh. Line 7 we compute the solution and line 8 the **global** error of the mesh.

While the global error is greater than $\tau$ (line 14), for each element we compute the **elemental** error (Line 16-18). If the elemental error is greater tha the tolerance, the we add the element to the list of the element to refine (Line 18-20)

**Remark** : The booleen variable analytic is here to specify that we know the exact solution of the problem. Moreover, we store the error in order to plot it for each step of the refinement.

### 2.2.3 Example

For the same tolerance than in the previous example, we need 2253 nodes and 4472 elements. Here the refined mesh :

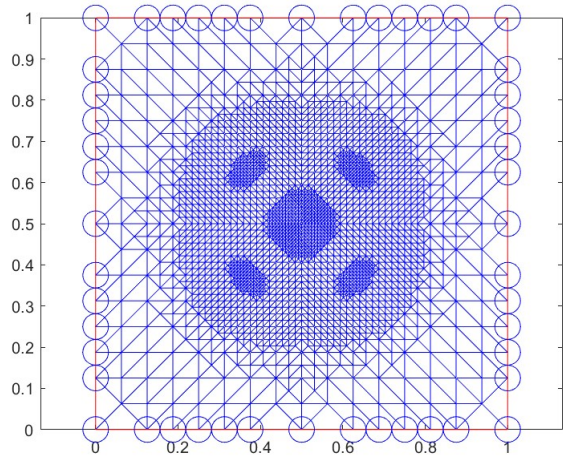


FIGURE 1 – Adaptative mesh

We can see that we need more refinement in the center. This coherent, because the solution is more sharper in center than in the sides, as we can see in the following graph.

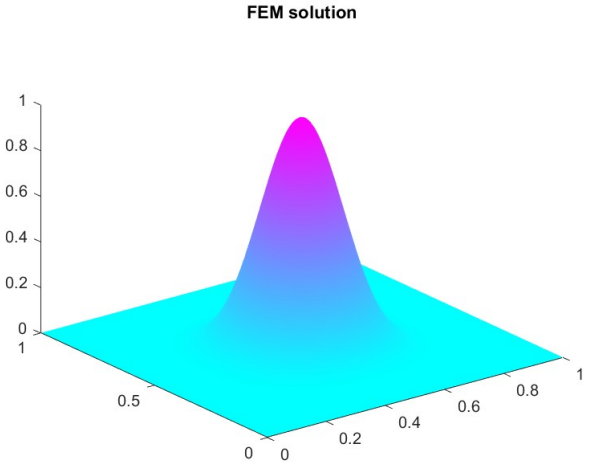


FIGURE 2 – Numerical solution

# 3 ZZ-estimator

In that part we are going to modify the previous algorithm, replacing the exact computation of the H1-semi-norm by it's estimator (ZZ-estimator). A code which compute the zz-estimator is given. Here the modified algorithm :

```
1       while error^2>tau
2          listElems=[];
3          error_elem=computeZZelementalErrors(u,mesh.X,mesh.T,theReferenceElement);
4          %maximum=max(error);
5          for i=1:size(error_elem)
6              if error_elem(i)^2>10^-5
```

```
7                   listElems=[listElems,i];
8               end
9           end
10          mesh=refineListElements(mesh,listElems,doPlot);
11          u=compute_sol(mesh.X,mesh.T,theReferenceElement);
12          error=compute_H1_semi_error(u,mesh.X,mesh.T,theReferenceElement);
13
14          error_semi=[error_semi,error];
15          h=[h,1/size(mesh.X,1)^2];
16          %disp(error);
17      end
18  end
```

Line 3 we compute the zz-estimator. And then, as before we refine an element if it's error is greater than the tolerance. In order to plot the error for each refinement, we store it line 14. As it is indicated, a good indicator for the size of the refined mesh is $\frac{1}{N^{1/d}}$ where d is the dimension and N the number of nodes of the mesh.

### 3.1 Example

If we take the same example as before, we need 2249 nodes and 4448 elements. It seems work as for the exact H1-norm.

## 4 Error

Here the error for each algorithm :

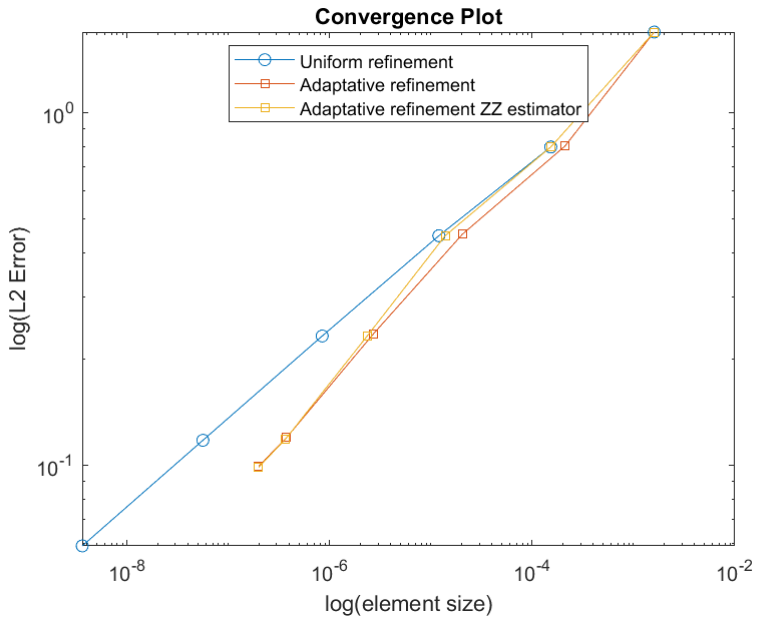


FIGURE 3 – Error in function of the size of the mesh

We c an see that the adaptive algorithm converges more faster than the uniform one. Also, it's need less elements, which is greater for computation time. Here, because it is a 2d-problem, it seems that the adaptative method is more efficient. However, in 3-d, the refinement of the mesh can take lot of time (because we have to compute the error for each elements).

# A Link to the code

All the code is available here :`https://github.com/leopaulbis/assignement_5_NMPDE`