Numerical Methods For PDE : Assignement 2

Leo-Paul BAUDET

28 octobre 2023

Table des matières

1 Weak form of the equation

The problem we are interesting to solve is the following (Laplace Equation) :

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

Let's v $\in H_0^1(\Omega)$ such that v=0 on $\partial\Omega$.We have :

$$\nabla.\nabla u = -f \Leftrightarrow v * \nabla.\nabla u = -f * v \text{ so } \int_\Omega v * \nabla.\nabla u d\Omega = -\int_\Omega f * v d\Omega \tag{2}$$

With the Divergence Theorem :

$$\int_\Omega v * \nabla.\nabla u d\Omega = -\int_\Omega \nabla v \nabla u d\Omega + \int_{\partial\Omega} v * u.n dS = -\int_\Omega \nabla v \nabla u d\Omega \text{ because v=0 on } \partial\Omega \tag{3}$$

Finally we obtain :

$$\int_\Omega \nabla u \nabla v d\Omega = \int_\Omega f * v d\Omega \tag{4}$$

and the associated weak form is :

$$\text{Find u} \in H^1(\Omega) \text{ st } u = u_d \text{ on } \partial\Omega \text{ st } \forall v \in H^1(\Omega) \text{ st } v = 0 \text{ on } \partial\Omega \text{ st : } \int_\Omega \nabla u \nabla v d\Omega = \int_\Omega f * v d\Omega \tag{5}$$

2 Discretization of the weak form

Ω is defined as rectangle ([0,1]*[0.1]). We are going to discretize this rectangle with triangles

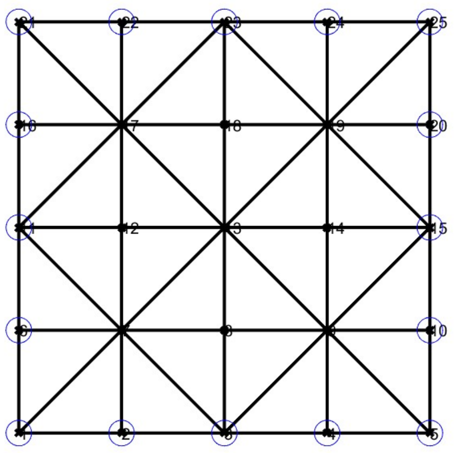


FIGURE 1 – Example of a mesh with triangles

Let's denote x_i the node number i of the discretization and Ω_e the element number e (triangle).

We are going to build an approximation u_h of the solution u st $u_h(x_i) \approx u(x_i)$ and $u_{h|_{[x_{j-1},x_j]}} \in \mathbb{P}_1([x_{j-1}, x_j])$ (degree of the method=1)

Let's define the nodal function st :

$$N_j(x_i) = \delta_{i,j} \tag{6}$$

Let's denote :

$$V_h = \left\{ u_h \in C^0(\Omega) \,\middle|\, u_{h|_{[x_{j-1},x_j]}} \in \mathbb{P}_1([x_{j-1}, x_j]) \forall j \in \{1, \ldots, n\} \right\} \tag{7}$$

and

$$V_{h,0} = \{u \in V_h \,|\, u = 0 \text{ on } \partial\Omega\} \tag{8}$$

We can show that $(N_j)_{0\leq j\leq n}$ is a base of V_h and that $(N_j)_{j\notin D}$ with $D = \{j|x_j \in \Gamma_D\}$ is a base of $V_{h,0}$ (here $\Gamma_D = \partial\Omega$)

Thus, we can rewrite the approximation u_h in that way :

$$u_h(x) = \sum_{j\notin D} u(x_j) * N_j(x) + \sum_{j\in D} u_d(x_j) * N_j(x) = \sum_{j\notin D} u(x_j) * N_j(x) + \Psi \tag{9}$$

In order to discretize the weak form, we are going to define $v_h = N_i(x)$ for $i \notin D$

Thus we have :

$$\int_\Omega \nabla u_h \nabla v_h d\Omega = \int_\Omega f * v_h d\Omega \tag{10}$$

Then, by linearity of all the operator (integral, derivatives), we have :

$$\sum_{j\notin D}(\int_\Omega \nabla N_j(x)\nabla N_i(x)d\Omega)u(x_i) = \int_\Omega f * N_i(x)d\Omega - \int_\Omega \nabla\Psi\nabla N_i d\Omega \tag{11}$$

$$\sum_{j\notin D} K_{i,j} * u_j = F_i \tag{12}$$

With

$$K_{i,j} = \int_\Omega \nabla N_i.\nabla N_j \, d\Omega \tag{13}$$

$$F_i = \int_\Omega N_i * f \, d\Omega - \int_\Omega \nabla\Psi\nabla N_i d\Omega \tag{14}$$

We can rewrite it as a system : $K * u = F$

3 Explanation of the code

3.1 Numerical procedure

After establishing the weak form of the Laplace equation and discretising it using the finite element method, we obtain the following formula $K * u = F$

In order to find the solution, we need to compute the value of K and the value of F and then solve the system.

It turns out that in calculating the coefficients of K, several integrals are identical (modulo the width of the element for example). From a numerical point of view, to optimise the calculation time and avoid calculating the same thing several times, we place ourselves on any element of the discretization, and calculate the integral on any element.

However, the geometric shape of the element is sometimes complex, which is why we use a reference element on which it is simpler to calculate the integral using an isoparametric change of variable, which goes from the reference element to the physical element, defined as follows (we denote by (i) the local index of the nodes on the reference element, ξ and η are the variable on Ω_{ref} and x and y on Ω) :

$$x(\xi,\eta) = \sum_{i=1}^{\#nodes} x_{(i)} N_i(\xi,\eta) \tag{15}$$

Moreover we can rewrite $K_{i,j}$ as follows on Ω_e :

$$k_{i,j} = \int_\Omega \nabla N_i.\nabla N_j \, d\Omega = \sum_e \int_{\Omega_e} \nabla N_i.\nabla N_j d\Omega_e \tag{16}$$

This is why define the elemental matrix K_e st :

$$K^e_{i,j} = \int_{\Omega_e} \nabla_{x,y} N_{(i)} . \nabla_{x,y} N_{(j)} \, d\Omega \tag{17}$$

Now we have to construct K from K_e. First of all, we have to do the correspondance between the local numbering in Ω_{ref} (here we have triangles so the local numbering is : (1),(2),(3)) and the global numbering (here we have n nodes). This is why we defined the connectivity matrix T. The coefficient $T_{i,j}$ is for the coefficient which corresponds to the global numbering of the i th element of local numbering j.

With that tool, we can construct the matrix K from K_e (assembly). However, with that way of construction of K (with K_e), it is difficult to combine the assembly of K and the selection of the nodes which are on the Dirichlet set (the coefficients of K correspond only to the factors in the discretised weak form which lie in front of the unknown nodal values of u).

This is why we first assemble the matrix without taking the Dirichlet conditions into account. We then select a sub-matrix to take account of the Dirichlet conditions.

This is why we consider the following discretized form (the same as before but whithout dirichlet condition ie whithout Ψ) :

$$\sum_{j=1}^{n} (\int_{\Omega} \nabla N_j(x) \nabla N_i(x) d\Omega) u(x_i) = \int_{\Omega} f * N_i(x) d\Omega \tag{18}$$

And then we have :

$$K_{i,j} = \int_{\Omega} \nabla_{x,y} N_j(x) \nabla_{x,y} N_i(x) d\Omega \tag{19}$$

$$F_i = \int_{\Omega} f * N_i(x) d\Omega \tag{20}$$

Then, the assembly is done in that way : For each element e, we calculate $k^e_{(i),(j)}$ for every i,j corresponding to a node number in the element, we use the e th row of the connectivity matrix to obtain the i and the j associated with the local numbering (i) and (j) (we need this to know the value of the nodal functions and to place the value in K). Next, we calculate the integral on the element and add its value to the corresponding place in K. (i th row and j th column of K). We are doing the same thing for F.

In order to have a second member corresponding to the K_e matrix, we define the elemental vector as follows(**we use the second member of the dsicretized form whithout Dirichlet condition**) :

$$F^e_i = \int_{\Omega_e} f * N_i \tag{21}$$

Once this has been done, we delete the lines of K corresponding to nodes located on the Dirichlet boundary. Next, we extract the columns (after the rows have been deleted) corresponding to the Dirichlet nodes. The final matrix K is obtained. For the second member, for each node on the Dirichlet boundary, we subtract the value of the solution at that node times the associated column without the rows deleted previously.

For example, if we have 5 nodes and two nodes on the Dirichlet boundary (node 1 and node 5).

$$k = \begin{pmatrix} 3 & 7 & -3 & 0 & 8 \\ 2 & 1 & 6 & 7 & 3 \\ -1 & 2 & 21 & 2 & 8 \\ 7 & 3 & 4 & 9 & 6 \\ -9 & -3 & 2 & 7 & 0 \end{pmatrix} \tag{22}$$

$$F = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \tag{23}$$

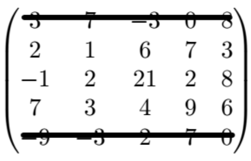


FIGURE 2 – Delete rows 1 and 5

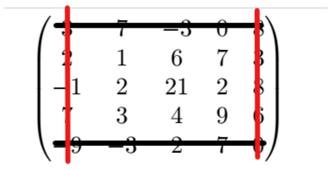


FIGURE 3 – Delete columns 1 and 5

Then we have K :

$$K = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 21 & 2 \\ 3 & 4 & 9 \end{pmatrix} \tag{24}$$

For the modified value of F, we select the columns of K corresponding to node 1 and 5 after deleting the rows corresponding to these two nodes :

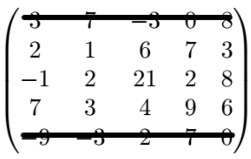


FIGURE 4 – Delete rows 1 and 5

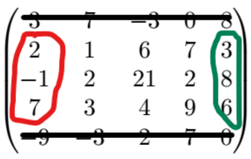


FIGURE 5 – Columns selected

Then the new value of F is :

$$\begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} - u_d(x_1) * \begin{pmatrix} 2 \\ -1 \\ 7 \end{pmatrix} - u_d(x_5) * \begin{pmatrix} 3 \\ 8 \\ 6 \end{pmatrix} \tag{25}$$

Now, with the change of variable mentioned above, we obtain the following formulas :

$$K^e_{i,j} = \int_{\Omega_{ref}} (J^{-1}\nabla_{\xi,\eta}N_i).(J^{-1}\nabla_{\xi,\eta}N_j)|J|d\Omega_{ref} \tag{26}$$

$$F^e_i = \int_{\Omega_{ref}} N_i * f|J|d\Omega_{ref} \tag{27}$$

With J the jacobian of the change of variable :

$$J = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{pmatrix} \tag{28}$$

Finally, we are going to use a numerical quadrature in order to compute the integral. Let w_k the weights and n_0 the number of integration point and z_g the integration points. Then we obtain the following formula :

$$K^e_{i,j} = \sum_{g=1}^{n_0} w_g * (J^{-1}(z_g)\nabla_{\xi,\eta}N_i(z_g).(J^{-1}(z_g)\nabla_{\xi,\eta}N_j(z_g)|J(z_g)| \tag{29}$$

$$f^e_i = \sum_{g=1}^{n_0} w_g * N_i(z_g) * f(x(z_g)) * |J(z_g)| \tag{30}$$

In the following part, we will explain how the matlab code work to compute K, f and solve the system.

3.2 Preprocess

In order to compute the solution numerically, we need to define a discretization of the space(define a reference element, a mesh with physical element, the domaine Ω and and the nodal function associated to the reference element). We are going to use the discretization explained below (with triangle and degree=1)

3.2.1 The reference element

First of all, in order to apply the finite element method, we need to define a reference element : this is the aim of the following function

```
function theReferenceElement = createReferenceElemeny(degree,typeOfElement)
%theReferenceElement = createReferenceElement(degree,typeOfElement)
%Creates a struct with the information of the reference element, for
%triangle discretizations

switch degree
    case 1
        if typeOfElement==1 %TRI
            z = [0.5 0; 0 0.5; 0.5 0.5];
            w = [1 1 1]/6;
            xi = z(:,1); eta=z(:,2);
            N = [1-xi-eta,   xi,   eta];
            Nxi  = [-ones(size(xi))   ones(size(xi))    zeros(size(xi))];
            Neta = [-ones(size(xi))   zeros(size(xi))   ones(size(xi))];
            nodesCoord = [0,0;1,0;0,1];
        else
            error('Element not implemented yet')
        end
    otherwise
        error('Element not implemented yet')
end
if typeOfElement==1
    typeElement='TRI';
else
    typeElement='QUA';
end
theReferenceElement=struct('IPweights',w,'IPcoord',z,'N',N,'Nxi',Nxi,'Neta',Neta,'type
```

Here we have the code which set a triangular reference element. The function return a structure with different fields :

— z contains the coordinate of the Gauss integration point in the element.
— w contains the coordinate of the weight of the quadrature formula
— N the values of the nodals functions at the Gauss points
— Nxi the derivatives of the nodals functions with respect to ξ at the integration point
— Neta the derivatives of the nodals functions with respect to η at the integration point
— nodesCoord the coordinates of the node in the element

The reason why we are evaluating the nodals functions and it's derivatives at the Gauss point is because, in the quadrature formula (25) we only need these values to compute the sum.

3.2.2 CreateMesh

Then we need to create a mesh : here we want to discretize a rectangle ([0,1]*[0,1]) with triangles. This is what the following function CreateMesh does :

```
figure(1), drawReferenceElement(theReferenceElement);
%Mesh: regular mesh in a rectangular domain [0,1]x[0,1]
```

```
[X,T] = CreateMesh(1,nOfElementNodes,[0,1,0,1],3,3);
figure(2), clf
```

n0OfElementNode is the number of node per element, 1 is for triangle and 5,5 it's to have 5 nodes on the both sides of the rectangle. It returns the coordinates of the nodes of the discretization, the connectivity matrix

3.2.3 Dirichlet Conditions

Here we select the node on the Dirchlet boundary and we calculate the value of the solution at these points

```
nodesCCD = find(abs(x)<tol|abs(x-1)<tol|abs(y)<tol|abs(y-1)<tol); %Nodes on the bounda
hold on, plot(x(nodesCCD),y(nodesCCD),'bo','MarkerSize',16); hold off
uCCD=DirichletValue(X(nodesCCD,:));

%System of equations (without BC)
```

— nodeccd are the node on Dirchlet boundary
— uCCD is the value of the solution at the dirichlet points

3.3 Computation of K and F

Now, we have defined the reference element and the nodes of the discretization. Now, we are going to compute K with the method explain in the Numerical procedure section.

3.3.1 ComputeSystemLaplace

The computeSytemLaplace function is doing assembly of the matrix K and of the vector F.

```
1  function [K,f]=computeSystemLaplace(X,T,theReferenceElement,sourceTermFunction)
2  
3  IPweights = theReferenceElement.IPweights;
4  IPcoord = theReferenceElement.IPcoord;
5  N=theReferenceElement.N;%function evaluated on the gauss point
6  Nxi=theReferenceElement.Nxi; %dérivative with respect to xi on the gauss point
7  Neta=theReferenceElement.Neta;%same with respect to eta
8  
9  %number of nodes =number of row of the matrix which contains the coordinate of the no
10 nOfNodes = size(X,1);
11 nOfElements = size(T,1);%number of element=nb of row in the connectivity matrix
12 
13 %initialize a sparse matrix of size n0nodes*n0nodes whith room for up to 9*nonodes
14 K=spalloc(nOfNodes,nOfNodes,9*nOfNodes);
15 f=zeros(nOfNodes,1);%initialise the vector of the second member
16 
17 %Loop in elements
18 for i=1:nOfElements
19     Te=T(i,:); %index of the nodes in the element
20     Xe=X(Te,:); %coordinates of the nodes in the element
21     [Ke,fe]=elementalComputations(Xe,IPcoord,IPweights,N,Nxi,Neta,sourceTermFunction)
22     K(Te,Te)=K(Te,Te)+Ke; %assembly of elemental matrix
23     f(Te) = f(Te) + fe;
24     %figure(11), spy(K), disp('Press any key to continue'), pause
25 end
```

The lines 3 to 7 are here to recover data in order to compute each value of $K^e_{(i),(j)}$ and $F^e_{(i)}$ (quadrature formula)

The number of element correspond to the the number of rows of the connectivity matrix and the number of nodes is the number of rows of the matrix which contains the coordinate of the nodes.

Then for each element (loop for), we compute the $K^e_{(i),(j)}$ and the $F^e_{(i)}$. In the line 18 and 19, we are using the connectivity matrix in order to know the coordinate of the nodes which constitutes the element e. Then we have all the information to compute the integrals which defines K_e and F^e (the coordinates of the element's nodes, the weights, the integration points,the derivative to construct the jacobian and the source term function : see the quadrature formulas). This is what the function ElementalComputation does (computation of K_e anf F^e. Then the lines 21 and 22 are doing the assembly of K (we add in K the value of the integral computed for (i) and (j) to the corresponding i and j coefficient in K)and the assembly of F (add the value computed for (i) to the corresponding i in F).

3.3.2 ElementalComputation

Here, the aim of this function is to compute these quadrature formula :

$$K^e_{i,j} = \sum_{g=1}^{n_0} w_g * (J^{-1}(z_g)\nabla_{\xi,\eta}N_i(z_g).(J^{-1}(z_g)\nabla_{\xi,\eta}N_j(z_g)|J(z_g)| \tag{31}$$

$$f^e_i = \sum_{g=1}^{n_0} w_g * N_i(z_g) * f(x(z_g)) * |J(z_g)| \tag{32}$$

```
1  function [Ke,fe]=elementalComputations(Xe,IPcoord,IPweights,N,Nxi,Neta,sourceTermFunct
2  
3  nnodes = size(Xe,1);
4  Ke=zeros(nnodes);
5  fe=zeros(nnodes,1);
6  xe = Xe(:,1); ye = Xe(:,2);
7  %Bucle en punts d integraci
8  for k=1:length(IPweights)
9      Nk=N(k,:);
10     Nkxi=Nxi(k,:);
11     Nketa=Neta(k,:);
12     xk = Nk*Xe; %map the gauss point to the physical element=isoparametric transformat
13     %Jacobia
14     J = [Nkxi*xe Nkxi*ye;Nketa*xe Nketa*ye];%jacobienne du changement isoparametric
15     % Derivadas de las funciones de forma respecto a (x,y)
16     Nkxy = J\[Nkxi;Nketa];
17     Nkx = Nkxy(1,:); Nky = Nkxy(2,:);
18     %diferencial de volum
19     dxy=IPweights(k)*det(J);
20     Gk=Nkxy;
21     Ke = Ke + Gk'*Gk*dxy;%fait le produit des deux gros trucs dans l'intégrale
22     fe = fe + sourceTermFunction(xk)*Nk'*dxy;
23 end
```

First we recover the number of nodes in the element(line 3). Then we create the matrix K_e and the vector F (line 4-5)(dimension : number of nodes in e*number of nodes in e for K).

We realize the sum on the integration point(loop for line 8). Then we compute the nodal function at the integration point(line 9) and it's derivative (line 10-11) in order to compute the jacobian.

Line 12 we are doing the isoparametric change of variable (defined in (15))(Nodal function in the reference element*the coordinate in the reference element).We need this value to compute the value of the source term in the real element. Then we compute the jacobian of this change of variable(see formula (24))(line 13).

Line 16 we compute $J^{-1}(z_g)*\nabla N_{j_{\xi,\eta}}(z_g)$ and line 19, we compute $w_g*|J(z_g)|$. Finally, line 21 and 22 we compute $w_g*(J^{-1}(z_g)\nabla_{\xi,\eta}N_i(z_g).(J^{-1}(z_g)\nabla_{\xi,\eta}N_j(z_g)|J(z_g)|$ and $w_g*N_i(z_g)*f(x(z_g))*|J(z_g)|$

3.3.3 Dirichlet Boundary conditions

Now we have computed K and F without taking into account the Dirichlet boundary conditions (as explained in the numerical procedure section). We have to select the rows and the colomns of K and modify the value of F. This is the aim of this part :

```
1       f = f(unknowns)-K(unknowns,nodesCCD)*uCCD;
2       K=K(unknowns,unknowns);
3       %System solution
4       sol=K\f;
5       %Nodal values: system solution and Dirichlet values
6       u = zeros(size(X,1),1);
7       u(unknowns) = sol; u(nodesCCD) = uCCD;
8   else %LagrangeMultipliers
9       nOfDirichletDOF = length(uCCD); nOfDOF = size(K,1);
10      A = spalloc(nOfDirichletDOF,nOfDOF,nOfDirichletDOF);
11      A(:,nodesCCD) = eye(nOfDirichletDOF);
12      b = uCCD;
```

In a part before, we have selected the nodes on the Dirichlet boundary. As explained before, we want first to delete the rows of K corresponding to the nodes on the Dirchlet boundary. This is why in line 5, we recover the nodes that are not in the Dirichlet boundary. With that we delete the rows and columns associated to these nodes in line7

$$\begin{pmatrix} \cancel{3} & \cancel{7} & \cancel{-3} & \cancel{0} & \cancel{8} \\ 2 & 1 & 6 & 7 & 3 \\ -1 & 2 & 21 & 2 & 8 \\ 7 & 3 & 4 & 9 & 6 \\ \cancel{-9} & \cancel{-3} & \cancel{2} & \cancel{7} & \cancel{0} \end{pmatrix}$$

FIGURE 6 – What does the instruction K(unknows,unknows)

Line 6,as explained in the numerical procedure, we compute the modified value of F : we have K without the rows of the nodes on the Dirichlet boundary. Then, we delete the rows of F which corresponds to the row deleted in K (instruction f(unknows))

$$\begin{pmatrix} \cancel{1} \\ 2 \\ 3 \\ 4 \\ \cancel{5} \end{pmatrix}$$

FIGURE 7 – What does the instruction f(unknows)

Then we substract the value of the solution at the Dirichlet nodes times the columns associated to the dirichlet nodes that are in K after deleting the rows(instruction -K(unknows,nodesCcd).

$$-u_d(x_1)*\begin{pmatrix} 2 \\ -1 \\ 7 \end{pmatrix} - u_d(x_5)*\begin{pmatrix} 3 \\ 8 \\ 6 \end{pmatrix}$$

FIGURE 8 – What does the instruction -K(unknows,nodesCcd)

Once we have the correct matrix K and vector F, we compute the system solution line 9 and line 12, we construct the real solution with the dirichlet nodes and the other nodes find with the finite element method.

3.4 Postprocess

```
%Comparison with analytical solution
figure(5)
[Xfine,Tfine] = CreateMesh(1,nOfElementNodes,[0,1,0,1],41,41);
PlotNodalField(analytical(Xfine),Xfine,Tfine), title('Analytical solution')

L2error=computeL2error(u,X,T,theReferenceElement);
H1error=computeH1error(u,X,T,theReferenceElement);
disp(L2error);


%PLOT OF THE L2 CONVERGENCE

%Reference element
```

Here we plot the solution and compute the L_2 error, in order to see if our code works.

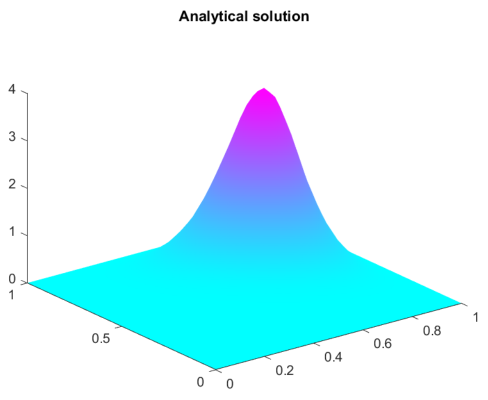


FIGURE 9 – Analytical solution

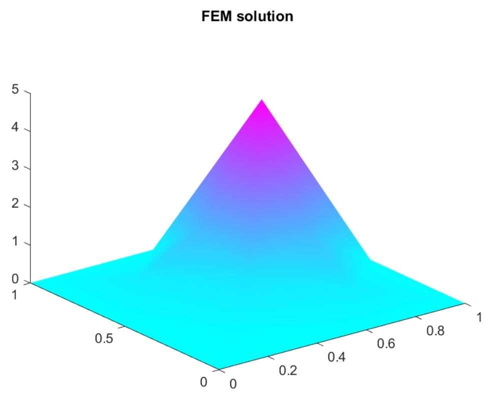


FIGURE 10 – FEM solution

3.4.1 Computation of the L2 error

Here we want to compute the L2 error defines as follows :

$$\|u - u^h\|_{L2} = (\int_{\Omega} |u - u^h|^2(x,y)dxdy)^{\frac{1}{2}} \tag{33}$$

However, for the same reason as before, we are going to compute the error using the reference element. First of all, we can rewrite the error as follows (with l the number of element) :

$$\|u - u^h\|_{L2}^2 = \sum_{e=1}^{l} \int_{\Omega_e - Physic} |u - u^h|^2(x,y)dxdy \tag{34}$$

Now with the isoparametric change of variable(with n the number of nodes in the element e) :

$$\|u-u^h\|_{L2}^2 = \sum_{e=1}^{l} \int_{\Omega_e - Ref} |u-u^h|^2(\sum_{i=1}^{n} N_i(\xi,\eta)x_i, \sum_{i=1}^{n} N_i(\xi,\eta)y_i)|J|d\xi d\eta = \int_{\Omega_e - Ref} |u-u^h|^2(x(\xi,\eta), y(\xi,\eta))|J|d\xi d\eta \tag{35}$$

With J the associated Jacobian of the change of variable and the N_i the nodals functions in the reference element. And then, we can apply a quadrature formula in order to compute this integral. Let w_k the weight of the quadrature formula and the integration points $((\xi_k, \eta_k) = z_k)$(we take α integration point) :

$$\|u - u^h\|_{L2}^2 = \sum_{e=1}^{l} \sum_{k=1}^{\alpha} |u - u^h|^2(x(z_k), y(z_k))|J| d\xi d\eta \tag{36}$$

So in order to compute the error we need to compute J, to evaluate the analytical solution to the integration points and then to recover the value of the numerical solution. Here the code which do that :

```
1  function errorL2=computeL2error(u,X,T,theReferenceElement)
2  
3  IPweights = theReferenceElement.IPweights;
4  IPcoord = theReferenceElement.IPcoord;
5  N=theReferenceElement.N;
6  Nxi=theReferenceElement.Nxi;
7  Neta=theReferenceElement.Neta;
8  
9  nOfElements = size(T,1);
10 errorL2 = 0;
11 %Loop in elements
12 for i=1:nOfElements
13     Te=T(i,:); Xe=X(Te,:);
14     xe = Xe(:,1); ye=Xe(:,2);
15     Ue=u(Te);
16     for g=1:length(IPweights)
17         N_g = N(g,:);
18         Nxi_g = Nxi(g,:);
19         Neta_g = Neta(g,:);
20         J = [Nxi_g*xe     Nxi_g*ye
21             Neta_g*xe  Neta_g*ye];
22         dvolu=IPweights(g)*det(J);
23         Xg = N_g*Xe;
24 
25         errorL2 = errorL2 + (analytical(Xg)-N_g*Ue)^2*dvolu;
26     end
27 end
28 errorL2 = sqrt(errorL2);
```

Line 12 we are iterating on the element (sum over e). Line 13-14 we recover the nodes on the element e and line 15 we take the value of the numerical solution for each nodes of the element e.

Then we iterate on the integration points (sum on the integration points). Line 17-20 we compute the jacobian of the isoparametric change of variable (evaluated on the integration points). Line 23, we are doing the change of variable. Line 22, we compute the $detJ * w_k$ in the sum.

4 Acurate Solution

In that section, we want to plot an acurate solution of the equation using the FEM with P1 elements.

First of all, we want to see if taking the element size=$h_e = \frac{1}{2}$ is suficient to have an accurate solution. We are searching the solution in the rectangle [0,1]*[0,1]. So if we want to have $h_e = \frac{1}{2}$, it is necessary to have 3 nodes on each side of the rectangle (evenly distributed). In order to do that, the only thing that we have modified is the following :replace the two last parameters of the function CreateMesh by 3 which means that we are taking 3 nodes on each sides of the rectangle.

```
[X,T] = CreateMesh(1,nOfElementNodes,[0,1,0,1],3,3);
```

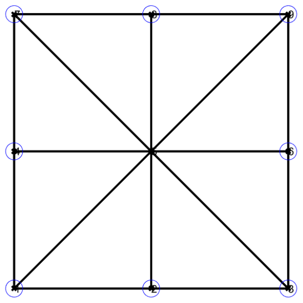


FIGURE 11 – Discretization with $h_e = \frac{1}{2}$

We obtain a L2 error of 0.580, which is not acurate at all. Indeed, we can see on the following plot that the solution does not seem as the analytical solution

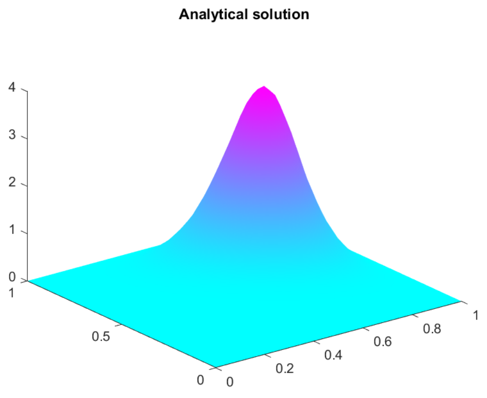


FIGURE 12 – Analytical solution

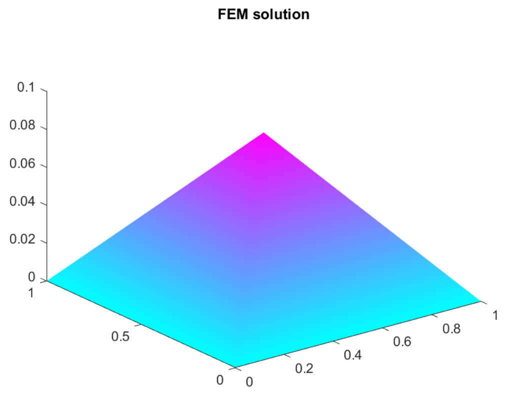


FIGURE 13 – FEM solution with $h_e = \frac{1}{2}$

If we take 40 nodes on each sides ($h_e = \frac{1}{39}$) we obtain a L2-error of 0.0097 which is more acurate. We can see that in the following plots :

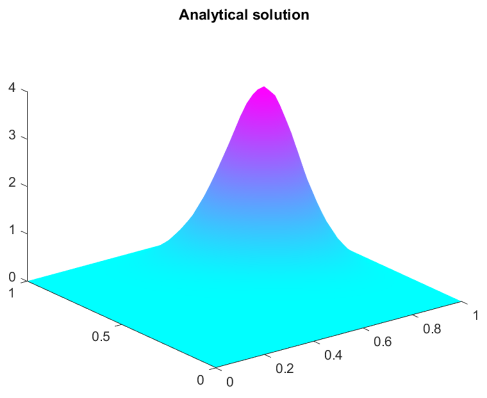


FIGURE 14 – Analytical solution

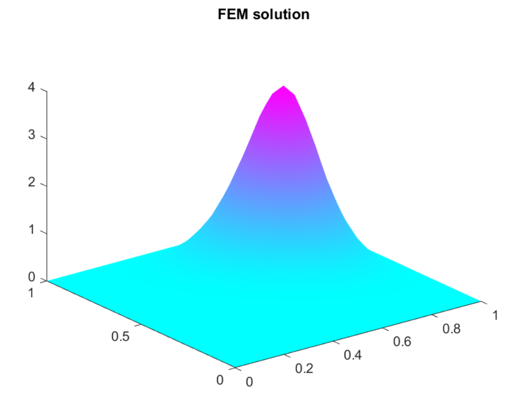


FIGURE 15 – FEM solution with $h_e = \frac{1}{39}$

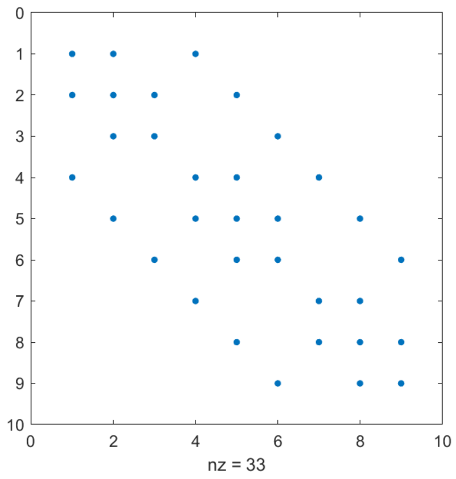


FIGURE 16 – Sparsity pattern of the matrix K with $h_e = \frac{1}{2}$

Let's recall that we have :

$$K_{i,j} = \int_{\Omega} \nabla N_i . \nabla N_j \, d\Omega = \sum_{e} \int_{\Omega_e} \nabla N_i . \nabla N_j \, d\Omega_e \tag{37}$$

This is why, $K_{i,j} \neq 0$ if and only if x_i and x_j belongs to the same element

This is clear that $K_{i,i} \neq 0$ (x_i and x_i belongs obviously to the same element).

5 Quadratic triangular element

In that section we are seeking to solve the laplace equation always with triangular elements but with a quadratic approximation of the solution. This is why we need 6 nodes per triangles. The only thing that will change is the number of nodes per element and so the number of shape functions.

We need to do two things : add a section in the function CreateReferenceElement to consider the quadratic case and to create a mesh with triangle with 6 nodes per triangles.

5.1 Creation of a quadratic element

Here we are going to add a section in the function CreateReferenceElement in order to use quadratic element. First of all let's considers the following reference element :

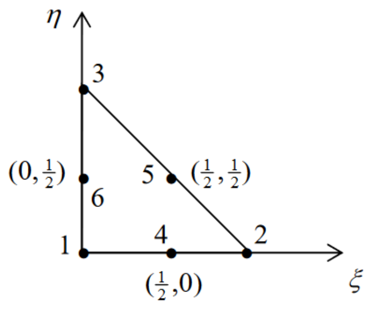


FIGURE 17 – Triangualr reference element for quadratic approximation

We are considering these six nodes : ((0,0),(1,0),(0,1),(1/2,0),(1/2,1/2),(0,1/2)). Here the associated shape function :

$$\begin{aligned}
N_1 &= (1-\xi-\eta)(1-2\xi-2\eta)\\
N_2 &= \xi(2\xi-1)\\
N_3 &= \eta(2\eta-1)\\
N_4 &= 4\xi(1-\xi-\eta)\\
N_5 &= 4\xi\eta\\
N_6 &= 4\eta(1-\xi-\eta)
\end{aligned}$$

We also need the partial derivatives of them :

$$\frac{\partial N_1}{\partial \xi}(\xi,\eta) = -3+4\eta+4\xi \text{ and } \frac{\partial N_1}{\partial \eta}(\xi,\eta) = -3+4\eta+4\xi \tag{38}$$

$$\frac{\partial N_2}{\partial \xi}(\xi,\eta) = 4\xi-1 \text{ and } \frac{\partial N_2}{\partial \eta}(\xi,\eta) = 0 \tag{39}$$

$$\frac{\partial N_3}{\partial \xi}(\xi,\eta) = 0 \text{ and } \frac{\partial N_3}{\partial \eta}(\xi,\eta) = 4\eta-1 \tag{40}$$

$$\frac{\partial N_4}{\partial \xi}(\xi,\eta) = 4-8\xi-4\eta \text{ and } \frac{\partial N_4}{\partial \eta}(\xi,\eta) = -4\xi \tag{41}$$

$$\frac{\partial N_5}{\partial \xi}(\xi,\eta) = 4\eta \text{ and } \frac{\partial N_5}{\partial \eta}(\xi,\eta) = 4\xi \tag{42}$$

$$\frac{\partial N_6}{\partial \xi}(\xi,\eta) = -4\eta \text{ and } \frac{\partial N_6}{\partial \eta}(\xi,\eta) = 4-4\xi-8\eta \tag{43}$$

Now the code which take in account these changements :

```
    case 2

         [z,w]=getTriangleQuadrature(5);
         xi = z(:,1); eta=z(:,2);

         N1=zeros(numel(xi),1);%contain the values of the nodal function N1
         % évaluated in all the integration points
         N2=zeros(numel(xi),1);
         N3=zeros(numel(xi),1);
         N5=zeros(numel(xi),1);
         N4=zeros(numel(xi),1);
         N6=zeros(numel(xi),1);

         N1xi=zeros(numel(xi),1);
         N2xi=zeros(numel(xi),1);
         N3xi=zeros(numel(xi),1);
         N4xi=zeros(numel(xi),1);
         N5xi=zeros(numel(xi),1);
         N6xi=zeros(numel(xi),1);

         N1eta=zeros(numel(xi),1);
         N2eta=zeros(numel(xi),1);
         N3eta=zeros(numel(xi),1);
         N4eta=zeros(numel(xi),1);
         N5eta=zeros(numel(xi),1);
         N6eta=zeros(numel(xi),1);

         for i=1:numel(xi)
            N1(i) = (1 - xi(i) - eta(i)) * (1 - 2 * xi(i) -2*eta(i));
            N2(i) = 2 * xi(i)^2- xi(i);
            N3(i) = 2 * eta(i)^2 - eta(i);
            N4(i) = 4 * xi(i) * (1 - xi(i) - eta(i));
            N5(i) = 4 * xi(i) * eta(i);
            N6(i) = 4 * eta(i) * (1 - xi(i) - eta(i));

            N1xi(i)=-3+4*xi(i)+4*eta(i);
            N2xi(i)=4*xi(i)-1;
            N4xi(i)=4-8*xi(i)-4*eta(i);
            N5xi(i)=4*eta(i);
            N6xi(i)=-4*eta(i);

            N1eta(i)=4*xi(i)+4*eta(i)-3;
            N3eta(i)=4*eta(i)-1;
            N4eta(i)=-4*xi(i);
            N5eta(i)=4*xi(i);
            N6eta(i)=4-4*xi(i)-8*eta(i);
         end

         N=horzcat(N1,N2,N3,N4,N5,N6); %nodal functions
         Nxi = [N1xi,N2xi,N3xi,N4xi,N5xi,N6xi];
         Neta = [N1eta,N2eta,N3eta,N4eta,N5eta,N6eta];

        nodesCoord = [0 0;1 0;0 1;1/2 0;1/2 1/2; 0 1/2];
```

First we are using the function getTriangleQuadrature in order to generate the integration point and the associetd weight (for a given order) in the reference element (the shape function have a degree two, so we need a higher order than before in order to be exact for the computation of the inetgrals). Here we have taken an order 5.

Then we are going to evaluate the shape function and it's derivatives at each integration point. The array N1 will contains the value of N1 in the integration points; N1xi it's derivative with respect to ξ and N1eta it's derivative with respect η.

5.2 Creation of a mesh for quadratic element

Here we just have to call the function CreateMesh with an odd number of nodes in each direcction (3 per sides of the element).

5.3 Quadratic element with $h_e = \frac{1}{2}$

Here we call the function createMesh as follows :

```
[X,T] = CreateMesh(1,nOfElementNodes,[0,1,0,1],3,3);
```

and we obtain :

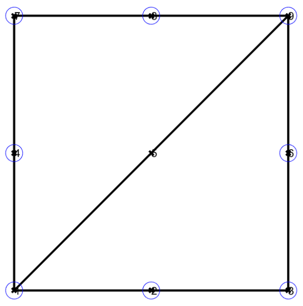


FIGURE 18 – Discretization with $h_e = \frac{1}{2}$

We obtain the following solution with an L2 error equals to 0.5564, which is better than in the case of linear element (logical, the L2 error is of the order of h_e^p).

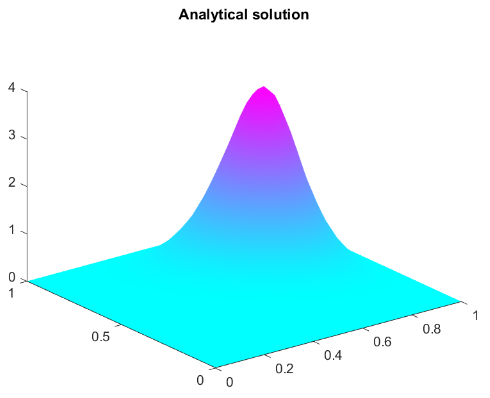


FIGURE 19 – Analytical solution

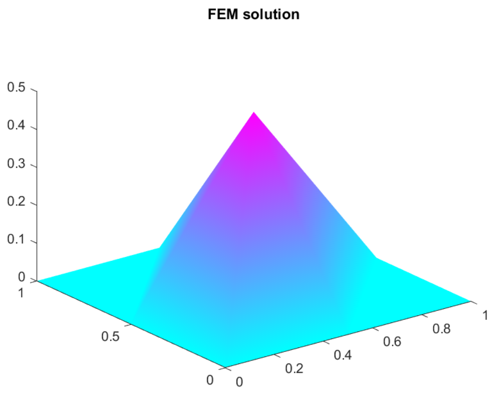


FIGURE 20 – FEM solution with $h_e = \frac{1}{2}$

6 Convergence plot

In that section we want to plot the L2 and H1 error in function of the number nodes taken. First of all we have to implement the computation of the H1 error for a given solution.

6.1 Computation of the H1 error

The H1 error is defined as follows :

$$\|u - u^h\|_{H1}^2 = \|u - u^h\|_{L2}^2 + \|\nabla u - \nabla u^h\|_{L2}^2 \tag{44}$$

We have already the code which compute $\|u - u^h\|_{L2}^2$. It remains to compute $\|\nabla u - \nabla u^h\|_{L2}^2$. The gradiant of the numerical solution is given by $\nabla_x N_i * X_g$ and $\nabla_y N_i * X_g$ where X_g are the integration points. We are using the same reasonement to compute the L2 error (using of the iso-paramaetric change of variable, ... etc). This is why we take the integration points on the reference element et the gradiant with repsect to ξ and η. Then we have to compute the gradiant of the analytic solution (using the partial derivatives).

The analytical solution is the following function :

$$f(x, y) = \frac{1}{2000} * h(x) * h(y) \text{ with } h(x) = x(1-x)^2 * exp(10x) \tag{45}$$

So we have :

$$\frac{\partial f}{\partial x}(x, y) = \frac{1}{2000} * h(y) * exp(10x) * (2x + 4x^2 - 16x^3 + 10x^4) \tag{46}$$

$$\frac{\partial f}{\partial x}(x, y) = \frac{1}{2000} * h(x) * exp(10y) * (2y + 4y^2 - 16y^3 + 10y^4) \tag{47}$$

Here the implementation

```

function y=analytic_deriv_x(X)
y=(1/2000)*h(X(:,2)).*v(X(:,1));

function y=v(x)
y=(2*x+4*x.^2-16*x.^3+10*x.^4).*exp(10*x);

function y=h(x)
y = (x.^2).*((1-x).^2).*exp(10*x);
```

```
function y=analytic_deriv_y(X)
y=(1/2000)*h(X(:,1)).*v(X(:,2));
```

Here the code to compute the H1error

```
1  function [H1err] = computeH1error(u, X, T, theReferenceElement)
2
3
4      IPweights = theReferenceElement.IPweights;
5      IPcoord = theReferenceElement.IPcoord;
6      N = theReferenceElement.N;
7      Nxi = theReferenceElement.Nxi;
8      Neta = theReferenceElement.Neta;
9
10     nOfElements = size(T, 1);
11     L2err = 0;
12     H1err = 0;
13
14     for i = 1:nOfElements
15         Te = T(i, :);
16         Xe = X(Te, :);
17         xe = Xe(:, 1);
18         ye = Xe(:, 2);
19         Ue = u(Te);
20
21         for g = 1:length(IPweights)
22             N_g = N(g, :);
23             Nxi_g = Nxi(g, :);
24             Neta_g = Neta(g, :);
25
26             J = [Nxi_g * xe, Nxi_g * ye; Neta_g * xe, Neta_g * ye];
27
28             grad_ref = [Nxi_g; Neta_g];
29
30             grad = J \ grad_ref;
31
32             Nx_g = grad(1, :);
33             Ny_g = grad(2, :);
34
35             dvolu = IPweights(g) * det(J);
36             Xg = N_g * Xe;
37
38             errorL2 = (analytical(Xg) - N_g * Ue) ^ 2;
39             L2err = L2err + errorL2 * dvolu;
40
41             errorH1 =(analytic_deriv_x(Xg) - Nx_g * Ue)^2 +
42             (analytic_deriv_y(Xg) - Ny_g * Ue)^2;
43             H1err = H1err +errorL2*dvolu+ errorH1 * dvolu;
44         end
45     end
46
47     H1err = sqrt(H1err);
48 end
```

The structure is the same as for the computation of the L2 error. Line 30-33 we compute the gradient matrix taht we will use to compute the gradiant of the numerical solution. Line 41 we compute $\|\nabla u - \nabla u^h\|^2_{L2}$ and line 43 $\|u - u^h\|^2_{H1}$

6.2 L2 convergence plot

Here we want to plot the L2 error in function of the element size. First of all we need to compute for different element size the L2 error. This is what is doing the function convergencel2.

6.2.1 convergence L2

```
1  function[node,error]=convergence_l2(i,num_dep,nOfElementNodes,theReferenceElement) %i
2      num=num_dep
3      error=zeros(1,i)
4      node=zeros(1,i)
5  
6      for k=1:i
7          [X,T] = CreateMesh(1,nOfElementNodes,[0,1,0,1],num,num)
8          node(1,k)=1/(num-1)
9          num=num+10*k
10 
11         x = X(:,1); y = X(:,2); tol=1.e-10;
12         nodesCCD = find(abs(x)<tol|abs(x-1)<tol|abs(y)<tol|abs(y-1)<tol);
13         uCCD=DirichletValue(X(nodesCCD,:));
14         [K,f]=computeSystemLaplace(X,T,theReferenceElement,@sourceTerm);
15 
16         unknowns= setdiff([1:size(X,1)],nodesCCD);
17         f = f(unknowns)-K(unknowns,nodesCCD)*uCCD;
18         K=K(unknowns,unknowns);%K_uu
19         %System solution
20         sol=K\f;
21         %Nodal values: system solution and Dirichlet values
22         u = zeros(size(X,1),1);
23         u(unknowns) = sol; u(nodesCCD) = uCCD;
24 
25         error(1,k)=computeL2error(u,X,T,theReferenceElement)
26     end
27 end
```

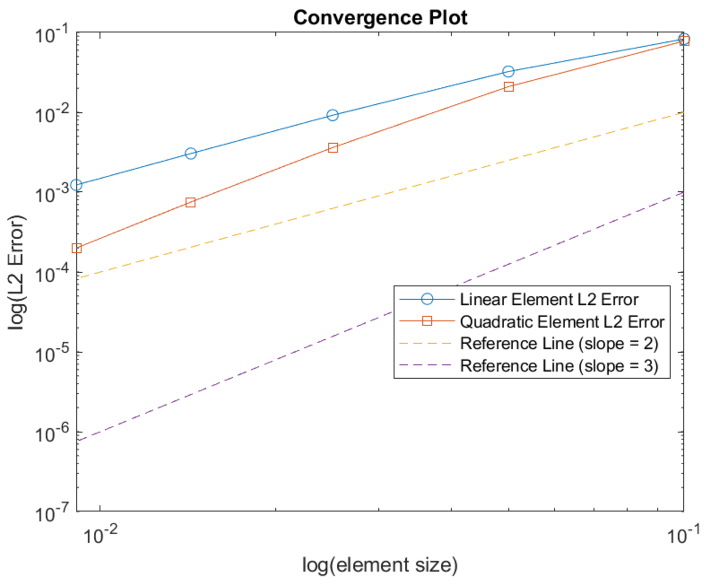
Num_{dep} is the number of nodes taken initially (in each direction). Then we are going to take num_{dep} nodes, then $num_{dep} + 10$ nodes, then $num_{dep} + 10 + 10^2$nodes ..etc. The parameter i is the maximal power of 10. Then for each solution, we store the L2 error in an array (line 25) and the element size in an other array (line 8).

6.2.2 Plot

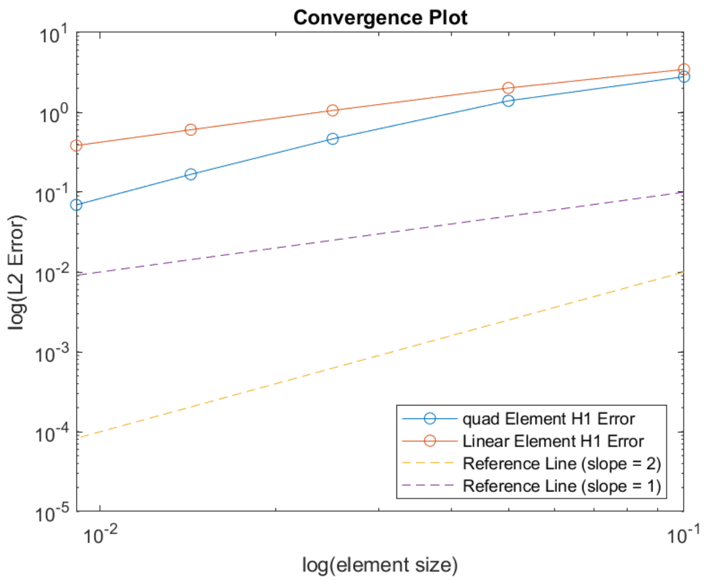
Using this funnction, we are going to plot the L2 error. Theorically, we know that $\|u - u^h\|^2_{L2} \leq c * h^{p+1}$ where p is the degree and h the element size. So we have :

$$log(\|u - u^h\|^2_{L2} = log(c) + (p + 1) * log(h) \tag{48}$$

This why if we plot $log(\|u - u^h\|^2_{L2})$ we must a have a straight line of slope p+1.



This is what we have. Then, we have implemented a similar function to convergeL2, in order to plot the L1error. (we have just change the line 25). Similarly as the L2 error, we can show that :$\|u - u^h\|^2_{L2} \leq c * h^p$. This why if we plot $log(\|u - u^h\|^2_{H1})$ we must a have a straight line of slope p. Then we obtain the following plot :



We can see that in the both case, the slope of the L2 error is of the order of p+1 and the slopz of the H1 error of the order of p.

A Link to the code

All the code is available here : `https://github.com/leopaulbis/assignement_2_NMPDE`