Numerical Methods For PDE : Assignement 3

Leo-Paul BAUDET

10 novembre 2023

Table des matières

1 Description of the problem

We are interested here in modelling the 2D movement of water in an orthotropic porous media Ω. This moovement is governed by the equation that imposes an incompressible flow q(x, y) :

$$\nabla q = 0 \tag{1}$$

For a linear steady flow, q is evaluated thanks to tha Darcy's law :

$$q = -K\nabla u \tag{2}$$

Where u is the piezometric level of the water, K=$\begin{pmatrix} K1 & 0 \\ 0 & K2 \end{pmatrix}$the hydraulic conductivity matrix. Here we will consider cosntant conductivity $K1 = 10^{-6}$m/s and $K2 = 10^{-7}$m/s

Here whe are interesting in the following domain Ω :

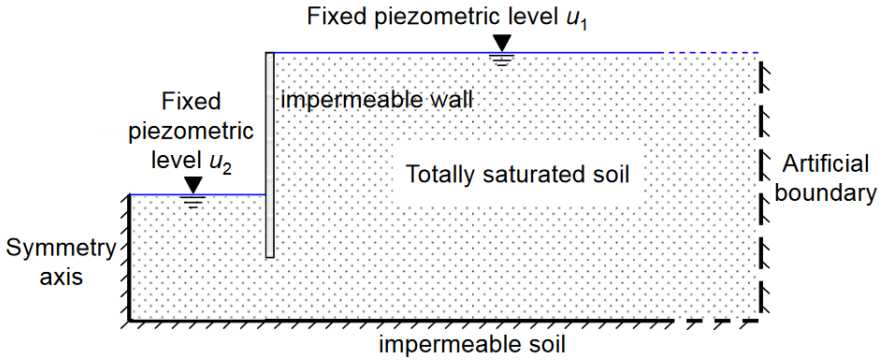


FIGURE 1 – Domain Ω

An artificial boundary is used to bound the computational domain. The symmetric, artificial and impermeable boundaries are modeled as $q * n = 0$, since the three conditions are equivalent to imposing that no flux goes though those boundaries. The two boundaries where the piezometric level is fixed are modeled by the Dirichlet condition $u(x, y) = h(y)$, where the height in this particular problem will be h(y) = y.

1.1 Strong form of the problem

From the previous part we have the following equation :

$$\nabla.(-K\nabla u) = 0 \tag{3}$$

and the following boundary sets :

$$\Gamma_n = \{\text{symmetric boundary, artificial boundary, impermeable boundary}\} \tag{4}$$

$$\Gamma_d = \{\text{fixed piezometric level of } u_1 \text{ and } u_2\}. \tag{5}$$

Then we apply Neumann boundary condition on Γ_n and Dirichlet boundary conditions (u(x,y)=h(y) where the height in this particular problem is h(y)=y) on Γ_d. Thus we obtain the following strong form :

$$\begin{cases} \nabla.(-K\nabla u) = 0 \text{ on } \Omega \\ -K\nabla u.n = 0 \text{ on } \Gamma_n \\ u(x, y) = u_d = y \text{ on } \Gamma_d \end{cases} \tag{6}$$

2 Weak form of the problem

2.1 Whithout lagrangian multipliers

Let $\in H_0^1(\Omega)$. Thus we have with an integration by part :

$$\int_\Omega v * \nabla.(-K\nabla u)d\Omega = 0 \iff \int_\Omega K * \nabla v * \nabla u d\Omega - \int_{\partial\Omega} v * K\nabla u.ndS = 0 \tag{7}$$

$$\iff \int_\Omega K * \nabla v.\nabla u d\Omega - \int_{\Gamma_n} v.q.ndS - \int_{\Gamma_d} K * v\nabla u.ndS \tag{8}$$

Because v=0 on Γ_d and $q.n$=0 on Γ_n we have :

$$\int_\Omega K * \nabla v.\nabla u d\Omega = 0 \tag{9}$$

Thus we obtain the following weak form :

Find u $\in H^1(\Omega)$ such that $u = u_d$ on Γ_d and $-K\nabla u.n = 0$ on Γ_n such that $\forall v \in H_0^1(\Omega)$:

$$\int_\Omega K * \nabla v.\nabla u d\Omega = 0 \tag{10}$$

2.1.1 Discretized weak form

Let (x_i, y_i) the nodes of the discretization of Ω. Let $D = \{j | x_j \in \Gamma_d\}$, let $v^h = N_i$ for $i \notin D$ and u^h such that :

$$u^h(x,y) = \sum_{j=1}^{n} u_j * N_j(x,y) = \sum_{j\in D} u_d(x_j,y_j) * N_j(x,y) + \sum_{j\notin D} u_j * N_j(x,y) = \Psi + \sum_{j\notin D} u_j * N_j(x,y) \tag{11}$$

With (9) :

$$\sum_{j\notin D} \int_\Omega K\nabla N_i \nabla N_j d\Omega u_j = -\int_\Omega K\nabla\Psi\nabla N_i d\Omega \tag{12}$$

We obtain the following system Ku=f with $K_{i,j} = \int_\Omega K\nabla N_i\nabla N_j d\Omega$ and $f_i = -\int_\Omega K\nabla\Psi\nabla N_i d\Omega$

2.2 With Lagrangian multipliers

The strong form can also be rewritten as minimization problem according to the paper written by Fernández-Méndez and Huerta (CMAME, 2004) :

$$u = \underset{\in u\in H^1(\Omega), u=u_d \text{ on } \Gamma_d}{\operatorname{argmin}} \Pi(v) \tag{13}$$

Where Π is the following energy :

$$\Pi(v) = \frac{1}{2}\int_\Omega \nabla v.\nabla u d\Omega \tag{14}$$

With the use of a Lagrange multiplier the minimization problem can be written as :

$$(u,\lambda) = arg \min_{v\in H^1(\Omega)} \max_{\gamma\in H^{-1/2}(\Gamma_d)} \Pi(v) + \int_{\Gamma_d} \gamma(v-u_d)d\Gamma \tag{15}$$

This leads to the following weak form with Lagrange multiplier : Find $u \in H^1(\Omega)$ and $\lambda \in H^{-1/2}(\Gamma_d)$ such that :

$$\int_\Omega \nabla u\nabla v d\Omega + \int_{\Gamma_d} v\lambda d\Gamma = 0, \forall v \in H^1(\Omega) \tag{16}$$

$$\int_{\Gamma_d} \gamma(u - u_d) d\Gamma = 0, \forall \gamma \in H^{-1/2}(\Gamma_d) \tag{17}$$

By a simple comparison between the both week forms, we have that the Lagrange multipliers represent the flux along the essential boundary (traction in mechanical problem).

Using the following discretization of $\lambda(x) = \sum_{i \in B} \lambda_i N_i^L(x)$ where $B = \{k | x_k \in \partial\Omega\}$, we obtain the following discretization :

$$\begin{pmatrix} K & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} u \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ b \end{pmatrix} \tag{18}$$

with

$$A_{i,j} = \int_{\Gamma_d} N_i^L N_j dS \text{ and } b_i = \int_{\Gamma_d} N_i^L u_d dS \tag{19}$$

3 Linear and quadratic quadrilateral elements

Here we want to solve the previous equation using quadrilateral elements (linear and quadratic). In order to do that we need to complete the function CreateReferenceElement. To check that our implementation we are going to test it on the Laplace equation (we know an analytical solution)

3.1 Linear quadrilateral element

First of all, let begin with the linear quadrilateral element :

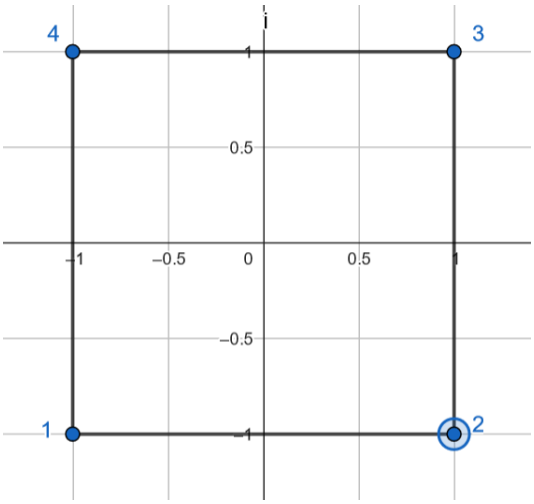


FIGURE 2 – Linear quadrilateral element

The nodal functions of this element can be obtained by the cartesian product beetween the nodal function of 1D linear element (straight line). Let's recall these nodal function :

$$N_1(x) = \frac{1}{2}(1 - x) \tag{20}$$

$$N_2(x) = \frac{1}{2}(x + 1) \tag{21}$$

Applying a cartesian product :

$$N_{1-quad-lin}(x, y) = N_1(x)N_2(y) = \frac{1}{4}(1 - x)(y + 1) \tag{22}$$

$$N_{2-quad-lin}(x, y) = N_1(x)N_1(y) = \frac{1}{4}(1 - x)(1 - y) \tag{23}$$

$$N_{3-quad-lin}(x, y) = N_2(x)N_1(y) = \frac{1}{4}(x + 1)(1 - y) \tag{24}$$

$$N_{4-quad-lin}(x,y) = N_2(x)N_2(y) = \frac{1}{4}(x+1)(y+1) \tag{25}$$

associated respectively to the node (-1,1), (-1,-1), (1,-1) and (1,1).

In similar way, we can generate the gauss point and the associated weight along the straight-line [-1,1] and with a tensor product generate the integration weight on the reference element. Here the code which do that :

```
        else
            [z,w]=gaussLegendre(4,-1,1);

            z=combvec(z,z)';  %(cartesian product)

            w=kron(w,w);%tensor product

           xi=z(:,1);
           eta=z(:,2);

           N1=zeros(numel(xi),1);
           N2=zeros(numel(xi),1);
           N3=zeros(numel(xi),1);
           N4=zeros(numel(xi),1);

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

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

           for i=1:numel(xi)
               N1(i)=1/4*(xi(i)-1)*(eta(i)-1);
               N2(i)=-1/4*(xi(i)+1)*(eta(i)-1);
               N3(i)=1/4*(xi(i)+1)*(eta(i)+1);
               N4(i)=-1/4*(xi(i)-1)*(eta(i)+1);

               N1xi(i)=1/4*(eta(i)-1);
               N2xi(i)=-1/4*(eta(i)-1);
               N3xi(i)=1/4*(eta(i)+1);
               N4xi(i)=-1/4*(eta(i)+1);

               N1eta(i)=1/4*(xi(i)-1);
               N2eta(i)=-1/4*(xi(i)+1);
               N3eta(i)=1/4*(xi(i)+1);
               N4eta(i)=-1/4*(xi(i)-1);
           end

           N=horzcat(N1,N2,N3,N4);
           Nxi = [N1xi,N2xi,N3xi,N4xi];
           Neta = [N1eta,N2eta,N3eta,N4eta];

           nodesCoord = [-1 -1;1 -1;1 1;1 -1;-1 1];
        end
```

Line 2 we generate the integration point and the associated weight along the segment [-1,1], then we use a tensor product (line 6) to obtain the weight associated to the points the cartesian product of the integration point (line 4).

Line 26 to 40, we compute the value of the shape function and it's derivative. at the inetgration points (in order to compute the quadrature formule later).

3.2 Test of the implementation on the Laplace equation

We obtain the following solution using the quadrilateral element and the following convergence plot :

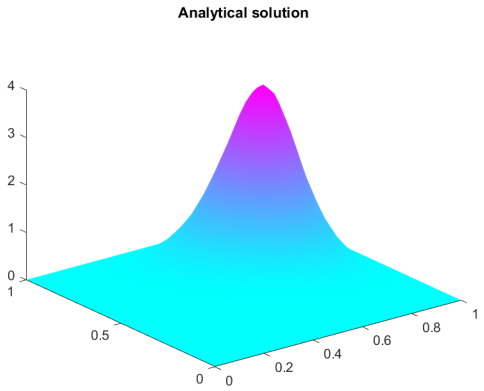


FIGURE 3 – Analytical solution

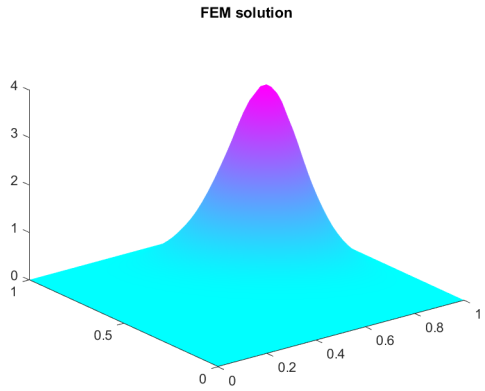


FIGURE 4 – FEM solution

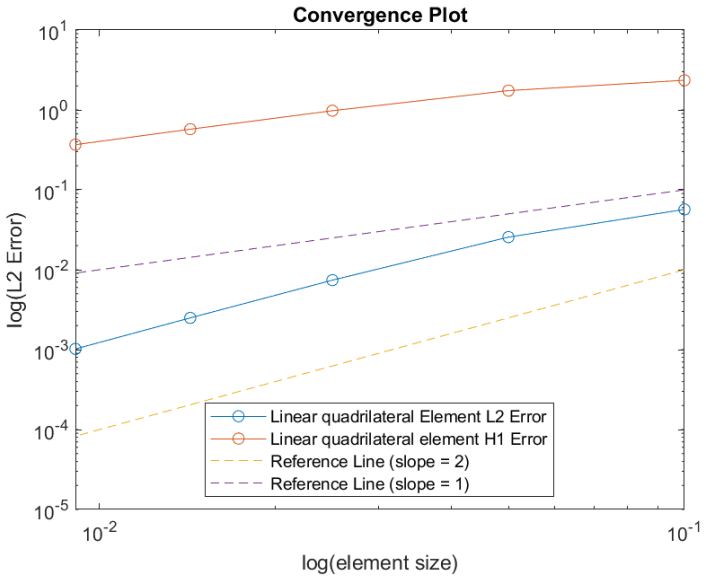
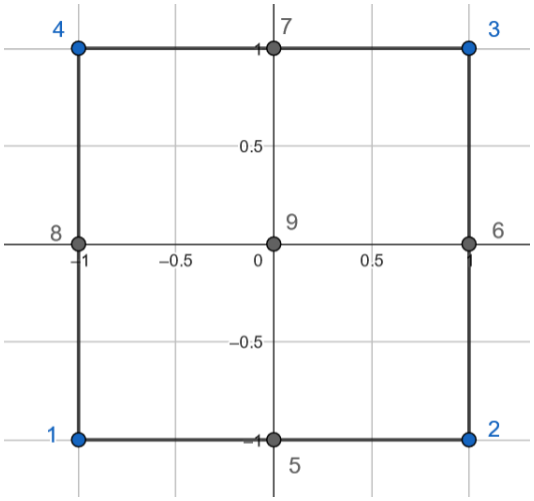


FIGURE 5 – Convergence plot linear quadrilateral

We clearly see that the L2 error as an order of h^{p+1} and the H1 error of h^p, so the linear quadrilateral element seems to be well implemented.

3.3 Quadratic quadrilateral element

Now we are considering this element :



In the same as before, we take the shape function of the 1D quadratic element and doing a tensor product of them. Let's recall the shape function of 1D quadratic element :

$$N_1(x) = \frac{1}{2}x(x-1) \tag{26}$$

$$N_2(x) = \frac{1}{2}x(x+1) \tag{27}$$

$$N_3(x) = (1-x)(x+1) \tag{28}$$

We obtain the following shape function for quadratic quadrilateral element :

$$N_1(x,y) = \frac{1}{4}x(x-1)(y-1)y \tag{29}$$

$$N_2(x,y) = \frac{1}{4}x(x-1)(y+1)y \tag{30}$$

$$N_3(x,y) = \frac{1}{2}x(x-1)(y+1)(1-y) \tag{31}$$

$$N_4(x,y) = \frac{1}{4}x(x+1)(y-1)y \tag{32}$$

$$N_5(x,y) = \frac{1}{4}x(x+1)(y+1)y \tag{33}$$

$$N_6(x,y) = \frac{1}{2}x(x+1)(y+1)(1-y) \tag{34}$$

$$N_7(x,y) = \frac{1}{2}(1-x)(x+1)(y-1)y \tag{35}$$

$$N_8(x,y) = \frac{1}{2}(1-x)(x+1)(y+1)y \tag{36}$$

$$N_9(x,y) = (1-x)(x+1)(y+1)(1-y) \tag{37}$$

Then it is the same idea to obtain the point of the quadrature and we are not going to detail the code (given in annexe).

3.4 Test of the implementation on the Laplace

As before, we test it on the Laplace equation. We obtain the following solution and the following convergence plot ::

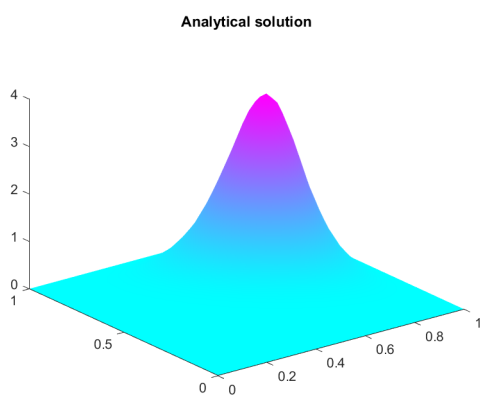


FIGURE 6 – Analytical solution

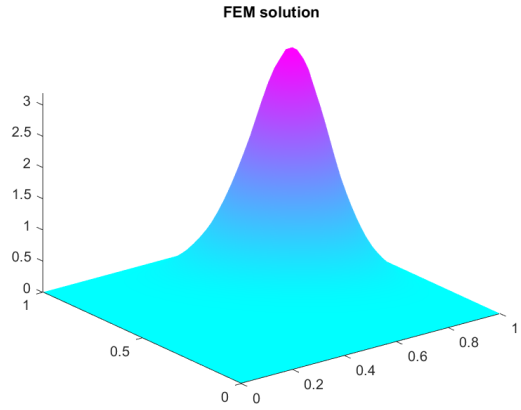


FIGURE 7 – FEM solution

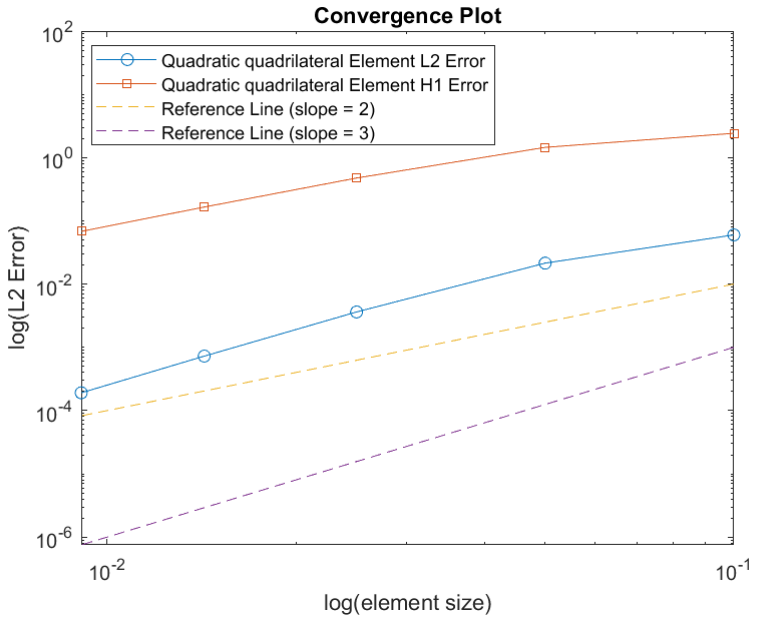


FIGURE 8 – Convergence plot quadratic quadrilateral

We clearly see that the L2 error as an order of h^{p+1} and the H1 error of h^p, so the quadratic quadrilateral element seems to be well implemented.

4 Numerical solving of the strong form

Here we want to solve the previous strong form (with length 5m=distance of teh artificial boundary) using linear and quadratic quadrilateral element. First of all we have to discretize with quadrilateral the figure 1.

4.1 Discretization of the space

In order to discretize the figure 1, we have created a function $mesh_pyzo$ to do that. This is just a function which load a file (provided) containing the discretization (connectivity matrix, nodes of the discretization) given a length and a type of element :

```
1  function [fileName]=mesh_pyzo(length,typeElem,orderElem,fineLevel)
2
3  %length    = '5'; %'5','10'%modify the length
4  %typeElem  = 'Q'; %'P','Q'
5  %orderElem = '2'; %'1','2'
6  %fineLevel = '2'; %'0','1','2','3'
7  orderElem=num2str(orderElem);
8  length=num2str(length);
9  fineLevel=num2str(fineLevel);
10 switch typeElem
11     case 'P'
12         geomElemType = 'triangle';
13         binElemType = 1;
14     case 'Q'
15         geomElemType = 'quadrilateral';
16         binElemType = 0;
17     otherwise
18         error('Not existent element type')
19 end
20
21 fileName = ['meshes/' length 'zanja' typeElem orderElem '_' fineLevel '_readed.mat'];
22
23 end
```

Line 7-8 we transform the lenght and OrderElem which are int into str (in order to construct the name of the file). Line 20, we contruct the file name and in the main we load it.

4.2 Dirichlet condition

After the computation of the global matrix, we have to take in account the Dirichlet conditions (we have also modified the source term function=0 for the computation of the global matrix). There are two ways to do that : using an assembly method or using Lagrange multipliers.

4.2.1 With an assembly step

Here we want to select the row and the columns which correspond to the nodes on the dirichlet boundary, this is why we have first of all to identify these nodes (nodes on h1 and h2 level).

```
1 T_boun_h1 = Tb_h1(:,[1 end]);
2 X_boun_h1 = X(T_boun_h1(:),:);
3
4 T_boun_h2 = Tb_h2(:,[1 end]);
5 X_boun_h2 = X(T_boun_h2(:),:);
6
7 nodesCCD=find(ismember(X,X_boun_h1,'rows')|ismember(X,X_boun_h2,'rows'));%find the noo
8 uCCD=DirichletValue2(X(nodesCCD,:));
```

Line 1-2 we are selecting the nodes on the piezometric level 1 and line 4-5 on the piezometric level 2.

Then we are computing the value of the solution at these nodes (known by dirichlet). Line 7 we get the indices of these nodes in X and line 8 we compute the value of the solution. Then, we compute K and f as before and we obtain the following distribution : There are logically more

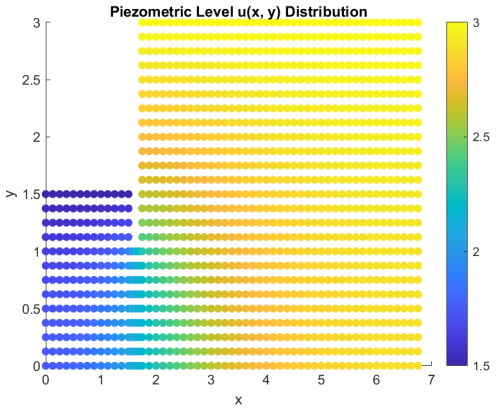


FIGURE 9 – Solution with linear quadrilatera

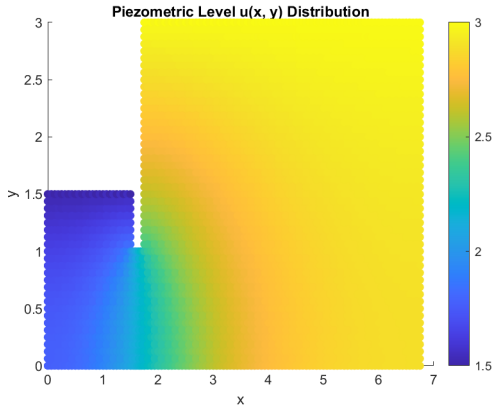


FIGURE 10 – Solution with quadratic quadrilatera

nodes with quadratic element than with linear element. However, the solution seems to be the same.

4.2.2 With Lagrange multipliers

Here are doing the same as before but using Lagrange multipliers to impose Dirichlet boundary condition. What we can see is that with two methods, we obtain exactly the same result.

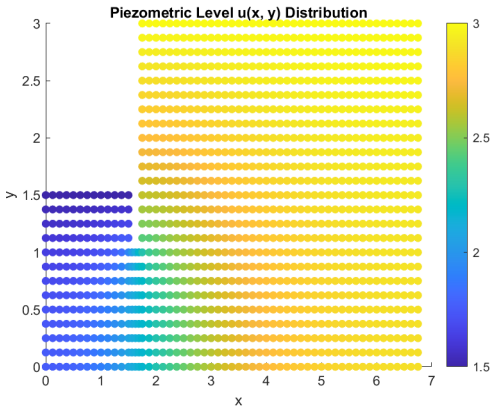


FIGURE 11 – Solution with linear quadrilatera

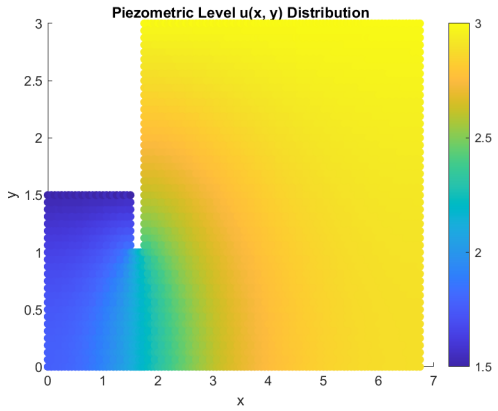


FIGURE 12 – Solution with quadratic quadrilatera

5 Impact of the disatnce of the artificial boundary

Here we are using Q2 quadrilateral element with Lagrange multiplier What we can see is that the value on the artificial boundary is "more constant" with L=10 than L=5. If we consider the boundary at the infinity, the value should be the same as at the top level of the domain. We can see that it is more the case when the boundary is at 10 m.

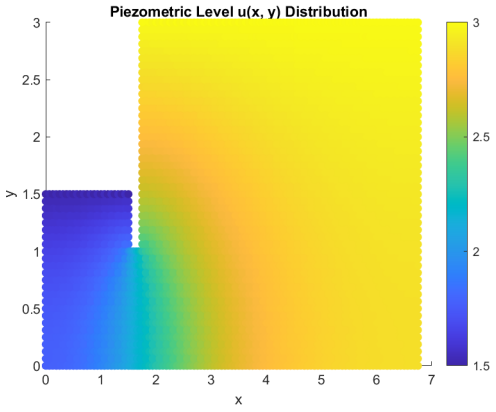


FIGURE 13 – Solution with L=5

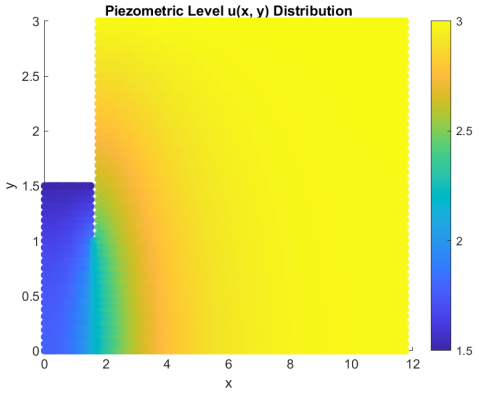


FIGURE 14 – Solution with L=10

6 Computation of the flux on the excavation surface

In order to that, we need to sum the lagrange multiplier which correpond to nodes on the excavation surface. In order to see the influence of the size of the mesh, we have created a function in order to iterate on the size of the mesh and sum the value of lambda on the right nodes : *mesh_influence_lambda*.

```
function[size_mesh,lambda_val]=mesh_influence_lambda(i,nOfElementNodes,theReferenceEle

    lambda_val=zeros(1,i);
    size_mesh=zeros(1,i);

    for k=1:i

        fileName=mesh_pyzo(10,'Q',degree,k-1); %creation of the associated mesh
        load(fileName);

        size_mesh(1,k)=k-1

        T_boun_h1 = Tb_h1(:,[1 end]);
        X_boun_h1 = X(T_boun_h1(:),:);

        T_boun_h2 = Tb_h2(:,[1 end]);
        X_boun_h2 = X(T_boun_h2(:),:);

        if bool

            T_boun = Tb_artificial(:,[1 end]);
            X_boun = X(T_boun(:),:);

            nodesCCD=find(ismember(X,X_boun_h1,'rows')|ismember(X,X_boun_h2,'rows')|is
        else
            nodesCCD=find(ismember(X,X_boun_h1,'rows')|ismember(X,X_boun_h2,'rows'));
        end
        uCCD=DirichletValue2(X(nodesCCD,:));

        [K,f]=computeSystemLaplace(X,T,theReferenceElement,@sourceTerm);

        nOfDirichletDOF = length(uCCD); nOfDOF = size(K,1);
        A = spalloc(nOfDirichletDOF,nOfDOF,nOfDirichletDOF);
```

```
34          A(:,nodesCCD) = eye(nOfDirichletDOF);
35          b = uCCD;
36          Ktot = [K A'; A spalloc(nOfDirichletDOF,nOfDirichletDOF,0)];
37          ftot = [f;b];
38          sol = Ktot\ftot;
39          u = sol(1:nOfDOF); lambda = sol(nOfDOF+1:end);
40
41          nodesCCD2=find(ismember(X,X_boun_h2,'rows'));
42
43          lambda2=lambda(1:numel(nodesCCD2));
44          lambda_val(1,k)=sum(lambda2);
45
46
47      end
48  end
```

The parameter i the max number of the fine level of the mesh and degree the degree of the element and bool a bollean to indicate if we put Dirichlet condition on the artificial boundary (denotes by Γ_{art}). For the other parameters, it is the same as before.

Line 3-4 we are created array in order to store the size of the mesh and the associated. Line 8-9, we are doing all the computation with length =10m.

Line 21-22 we impose Dirichlet boundary condition on the artificial boundary and line 43 we compute the flux on the excavation surface.

We obtain the following values

Flux on the excavation surface		
Fine level	Flux without Dirichlet on Γ_{art}	Flux with Dirichlet on the Γ_{art}
0	0.5665	0.5547
1	0.5867	0.5759
2	0.5988	0.5885
3	0.6053	0.5952

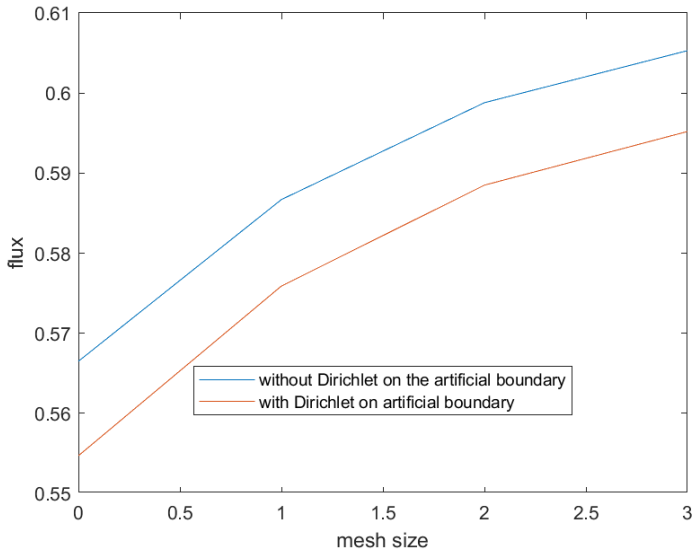


FIGURE 15 – Influence of the mesh size on the value of the flux

We can see that if we impose Dirichlet boundary coundition at the artificial boundary, the flux is a little bit smaller than in the other case. This is logical because in the first case, we impose that there is no flux accros $\Gamma_a rt$, so in the second case, there is a little perdition due to the flux accros this boundary.

A Link to the code

All the code is available here : `https://github.com/leopaulbis/assignement3_NMEDP`