UNIVERSITAT POLITÈCNICA DE CATALUNYA

Master on Numerical Methods in Engineering

HOMEWORK 1

Course: Programming for Engineers and Scientists

Name: Yonatan Lisne Luque Apaza

April 2017

1. Introduction

This report accompanies the finite element solver, developed in matlab, for the Poisson equation in two dimensions under a domain of [0 1] x [0 1] with an obstacle. The solver can resolve other domains by varying the ***GetBoundary.m*** function.

The general characteristics of the program are shown in addition to the convergence analysis for the homework meshes.

2. Finite Element for Poisson Equation in 2D

2.1.Poisson Equation

The nomenclature of the variables is the same in the program. The governing equation is given by:

$$\boldsymbol{\nabla}.(D\boldsymbol{\nabla}\phi) + Q = 0 \qquad in\ \ \Omega$$

$$\phi - \bar{\phi} = 0 \qquad on\ \ \Gamma_d$$

$$\boldsymbol{n}.\boldsymbol{q} = q_n + \alpha(\phi - \phi_q) \qquad on\ \ \Gamma_n$$

$$\boldsymbol{q} = -D\boldsymbol{\nabla}\phi$$

2.2.Weak form

$$\int_\Omega (\boldsymbol{\nabla}w).(D\boldsymbol{\nabla}\phi)\ d\Omega + \int_{\Gamma_n} w\alpha\phi\ d\Gamma = \int_\Omega wQ\ d\Omega - \int_{\Gamma_n} w\big(q_n - \alpha\phi_q\big)d\Gamma + \int_{\Gamma_d} (wD\boldsymbol{\nabla}\phi).\boldsymbol{n}\ d\Gamma$$

For Galerkin formulation:

$$w = N_i \qquad\qquad \boldsymbol{\nabla}w = \boldsymbol{\nabla}N_i$$

$$\phi = \sum_j^n N_j\phi_j = N_j\phi_j \qquad\qquad \boldsymbol{\nabla}\phi = \sum_j^n (\boldsymbol{\nabla}N_j)\phi_j = (\boldsymbol{\nabla}N_j)\phi_j$$

Is obtained:

$$\int_\Omega (\boldsymbol{\nabla}N_i).\big(D(\boldsymbol{\nabla}N_j)\phi_j\big)\ d\Omega + \int_{\Gamma_n} N_i\alpha N_j\phi_j\ d\Gamma = \int_\Omega N_iQ\ d\Omega - \int_{\Gamma_n} N_i\big(q_n - \alpha\phi_q\big)d\Gamma + r_{\Gamma_d}$$

For each element:

$$K_{ij}^{(e)} = \int_{\Omega^{(e)}} D\left(\boldsymbol{\nabla}N_i^{(e)}\right).\left(\boldsymbol{\nabla}N_j^{(e)}\right)\ d\Omega + \int_{\Gamma_n^{(e)}} \alpha N_i^{(e)}N_j^{(e)}\ d\Gamma$$

$$f_i^{(e)} = \int_{\Omega^{(e)}} N_i^{(e)}Q\ d\Omega - \int_{\Gamma_n^{(e)}} N_i^{(e)}\big(q_n - \alpha\phi_q\big)d\Gamma + r_{\Gamma_d}^{(e)}$$

2.3.Matrix form

Considering:

$$\boldsymbol{N}^{(e)} = \begin{bmatrix} N_1^{(e)} & N_2^{(e)} & \dots & N_{ne}^{(e)} \end{bmatrix}$$

$$\boldsymbol{X}^{(e)} = \begin{bmatrix} x_1^{(e)} & y_1^{(e)} \\ x_2^{(e)} & y_2^{(e)} \\ \vdots & \vdots \\ x_{ne}^{(e)} & y_{ne}^{(e)} \end{bmatrix} \qquad \boldsymbol{\phi}^{(e)} = \begin{bmatrix} \phi_1^{(e)} \\ \phi_2^{(e)} \\ \vdots \\ \phi_{ne}^{(e)} \end{bmatrix}$$

Is obtained:

$$[x \quad y]^{(e)} = \boldsymbol{N}^{(e)}\boldsymbol{X}^{(e)} \qquad \phi^{(e)} = \boldsymbol{N}^{(e)}\boldsymbol{\phi}^{(e)}$$

$$\boldsymbol{B}^{(e)} = \boldsymbol{\nabla}\boldsymbol{N}^{(e)} = \begin{bmatrix} \frac{dN_1^{(e)}}{dx} & \frac{dN_2^{(e)}}{dx} & \dots & \frac{dN_{ne}^{(e)}}{dx} \\ \frac{dN_1^{(e)}}{dy} & \frac{dN_2^{(e)}}{dy} & \dots & \frac{dN_{ne}^{(e)}}{dy} \end{bmatrix}$$

Introducing local coordinates:

$$\boldsymbol{B}_{\xi\eta}^{(e)} = \boldsymbol{\nabla}_{\xi\eta}\boldsymbol{N}^{(e)} = \begin{bmatrix} \frac{\partial N_1^{(e)}}{\partial \xi} & \frac{\partial N_2^{(e)}}{\partial \xi} & \dots & \frac{\partial N_{ne}^{(e)}}{\partial \xi} \\ \frac{\partial N_1^{(e)}}{\partial \eta} & \frac{\partial N_2^{(e)}}{\partial \eta} & \dots & \frac{\partial N_{ne}^{(e)}}{\partial \eta} \end{bmatrix}$$

$$\boldsymbol{B}_{\xi}^{(e)} = \boldsymbol{\nabla}_{\xi}\boldsymbol{N}^{(e)} = \begin{bmatrix} \frac{\partial N_1^{(e)}}{\partial \xi} & \frac{\partial N_2^{(e)}}{\partial \xi} & \dots & \frac{\partial N_{ne}^{(e)}}{\partial \xi} \end{bmatrix}$$

The Jacobian matrix for two and one dimension:

$$\boldsymbol{J}^{(e)} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \boldsymbol{B}_{\xi\eta}^{(e)}\boldsymbol{X}^{(e)}$$

$$\boldsymbol{J}_{1D}^{(e)} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \end{bmatrix} = \boldsymbol{B}_{\xi}^{(e)}\boldsymbol{X}^{(e)}$$

Is obtained:

$$\boldsymbol{B}^{(e)} = (\boldsymbol{J}^{(e)})^{-1}\boldsymbol{B}_{\xi\eta}^{(e)}$$

The matrix and vector in interior and boundary:

$$\boldsymbol{K}^{(e)} = \boldsymbol{K}_{\boldsymbol{\Omega}}^{(e)} + \boldsymbol{K}_{\Gamma_n}^{(e)}$$

$$\boldsymbol{f}^{(e)} = \boldsymbol{f}_{\boldsymbol{\Omega}}^{(e)} + \boldsymbol{f}_{\Gamma_n}^{(e)} + \boldsymbol{r}_{\Gamma_d}^{(e)}$$

Matrix form and quadrature integration with $\boldsymbol{z}_g$ point and w_g weights of each element:

$$\boldsymbol{K}_{\boldsymbol{\Omega}}^{(e)} = \int_{\Omega_{\xi\eta}^{(e)}} D\left(\boldsymbol{B}^{(e)}\right)^{\boldsymbol{T}}\left(\boldsymbol{B}^{(e)}\right)\left|\boldsymbol{J}^{(e)}\right| d\Omega_{\xi\eta} = \sum_{g=1}^{n_g} w_g\, D_{(\boldsymbol{z}_g)}\left(\boldsymbol{B}^{(e)}{}_{(\boldsymbol{z}_g)}\right)^{\boldsymbol{T}}\left(\boldsymbol{B}^{(e)}{}_{(\boldsymbol{z}_g)}\right)\left|\boldsymbol{J}^{(e)}{}_{(\boldsymbol{z}_g)}\right|$$

$$\boldsymbol{K}_{\Gamma_n}^{(e)} = \int_{\Gamma_{n_\xi}^{(e)}} \alpha(\boldsymbol{N}^{(e)})^{\boldsymbol{T}}(\boldsymbol{N}^{(e)})\left|\boldsymbol{J}_{1D}^{(e)}\right| d\xi = \sum_{g=1}^{n_g} w_g \alpha_{(\boldsymbol{z}_g)}(\boldsymbol{N}^{(e)}{}_{(\boldsymbol{z}_g)})^{\boldsymbol{T}}(\boldsymbol{N}^{(e)}{}_{(\boldsymbol{z}_g)})\left|\boldsymbol{J}_{1D}^{(e)}{}_{(\boldsymbol{z}_g)}\right|$$

$$\boldsymbol{f}_{\boldsymbol{\Omega}}^{(e)} = \int_{\Omega_{\xi\eta}^{(e)}} (\boldsymbol{N}^{(e)})^{\boldsymbol{T}} Q\left|\boldsymbol{J}^{(e)}\right| d\Omega_{\xi\eta} = \sum_{g=1}^{n_g} w_g\,(\boldsymbol{N}^{(e)}{}_{(\boldsymbol{z}_g)})^{\boldsymbol{T}} Q_{(\boldsymbol{z}_g)}\left|\boldsymbol{J}^{(e)}{}_{(\boldsymbol{z}_g)}\right|$$

$$\boldsymbol{f}_{\Gamma_n}^{(e)} = -\int_{\Gamma_{n_\xi}^{(e)}} (\boldsymbol{N}^{(e)})^{\boldsymbol{T}}\left(q_n - \alpha\phi_q\right)\left|\boldsymbol{J}_{1D}^{(e)}\right| d\xi = \sum_{g=1}^{n_g} w_g\,(\boldsymbol{N}^{(e)}{}_{(\boldsymbol{z}_g)})^{\boldsymbol{T}}\left(q_{n_{(\boldsymbol{z}_g)}} - \alpha_{(\boldsymbol{z}_g)}\phi_{q_{(\boldsymbol{z}_g)}}\right)\left|\boldsymbol{J}_{1D}^{(e)}{}_{(\boldsymbol{z}_g)}\right|$$

Obtain vector $\boldsymbol{r}_{\Gamma_d}^{(e)}$ is not necessary. Because we will eliminate the equations where ϕ_j value are previously established, and replace this value in the rest of equations.

Finally $\boldsymbol{\nabla\phi}$ values are obtained as an average of the adjoining elements in each node.

$$(\boldsymbol{\nabla\phi})^{(\boldsymbol{e})} = \begin{bmatrix} \dfrac{d\phi}{dx} \\ \dfrac{d\phi}{dx} \end{bmatrix}^{(\boldsymbol{e})} = \boldsymbol{B}^{(e)}\boldsymbol{\phi}^{(e)} = (\boldsymbol{J}^{(e)})^{-\mathbf{1}}\boldsymbol{B}_{\xi\eta}^{(e)}\boldsymbol{\phi}^{(e)}$$

3. Matlab Solver

The program can be started in ***MainSolver.m*** where you will be asked to enter the names of the files containing the nodes and elements, and then ask if you want to set different conditions to the preset parameters, which match the homework:

$$\begin{cases} \triangle u = 0 & \text{in } \Omega, \\ \nabla u \cdot \mathbf{n} = -1 & \text{on } \Gamma_{\text{in}} = 0 \times (0,1), \\ \nabla u \cdot \mathbf{n} = 1 & \text{on } \Gamma_{\text{out}} = 1 \times (0,1), \\ \nabla u \cdot \mathbf{n} = 0 & \text{on } \partial\Omega \backslash (\Gamma_{\text{in}} \cup \Gamma_{\text{out}}), \\ u(0,0) = 0 \end{cases}$$

If you select the parameters by default, the program will start ***SolverDefault.m***, Which has a similar procedure if other parameters are established. The results will be generated with the name ***Solution.vtk***.

The ***HWsolver.m*** function solve all the meshes of the homework.

3.1.Standard solution process

All solutions follow the standard process Shown in *Figure 1*. It starts with reading the files containing the mesh, and then you get the boundary elements with the **GetBoundary.m**.

After obtaining all the information of the mesh and selecting the appropriate reference element, the global matrix is assembled with the **FEM_System.m** function.

Then, the Neumann conditions for each of the boundary of this type are applied through the **SetNeumanBoundary.m** function.

The last condition to apply is the Dirichlet type with the SetDirichletBoundary.m function.

Finally the linear system is solved and the solution is exported with **ToParaview.m** function**.**

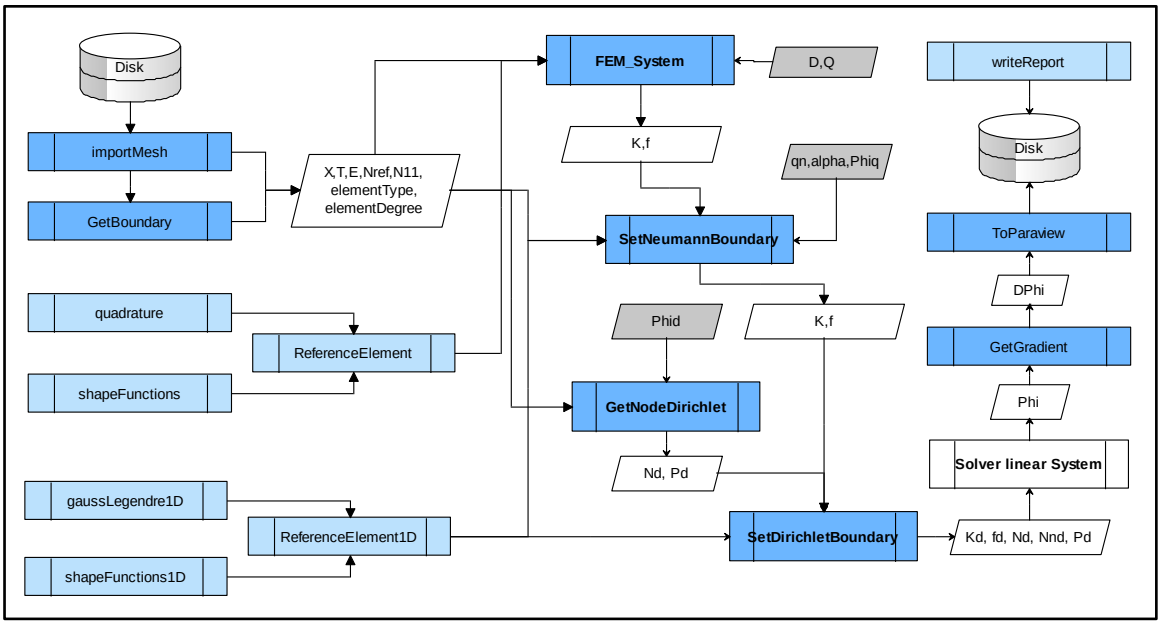

Figure 1: Standard solution process schema

3.2.Main functions

3.2.1. FEM_System

In this function we obtain the values of the global assembly of all the $\boldsymbol{K}_{\boldsymbol{\Omega}}^{(e)}$ matrices and $\boldsymbol{f}_{\boldsymbol{\Omega}}^{(e)}$ vectors, according to the matrix form presented.

FEM_System.m	
↖ K	Global Matrix of K
↖ f	Global Vector of f
↘ X	Nodal coordinates
↘ T	Connectivities
↘ referenceElement	Reference element for 2D
↘ D	D parameter (function of ‘x’ and ‘y’)
↘ Q	Q parameter (function of ‘x’ and ‘y’)

Table 1: Input and Output of function

```
K = zeros(nPt,nPt);
f = zeros(nPt,1);

for i=1:nElem
   Te=T(i,:);
   Xe=X(Te,:);
   Ke=zeros(nen,nen);
   fe=zeros(nen,1);
   for j=1:nOfGauss
      N_j=N(j,:);
      Nxi_j=Nxi(j,:);
      Neta_j=Neta(j,:);
      w_j=wgp(j);
      xg=N_j*Xe(:,1);
      yg=N_j*Xe(:,2);
      J_j=[Nxi_j ; Neta_j]*Xe;
      B_j=J_j\[Nxi_j ; Neta_j];
      Ke=Ke+B_j'*B_j*D(xg,yg)*w_j*det(J_j);
      fe=fe+N_j'*Q(xg,yg)*w_j*det(J_j);
   end
   K(Te,Te)=K(Te,Te)+Ke;
   f(Te)=f(Te)+fe;
```

Figure 2: FEM_System.m code

3.2.2. SetNeumannBoundary

This function adds $\boldsymbol{K}_{\Gamma_n}^{(e)}$ and $\boldsymbol{f}_{\Gamma_n}^{(e)}$ values of all the elements to the global assembly, according to the matrix form presented.

SetNeumannBoundary.m	
↖ K	Global Matrix of K
↖ f	Global Vector of f
↘ K	Global Matrix of K
↘ f	Global Vector of f
↘ X	Nodal coordinates
↘ E	Conectivities [nBound n1' n2' .. ne']
↘ nBoundary	Numero of boundary to set
↘ referenceElement	Reference element for 1D
↘ qn	qn parameter (function of 'x' and 'y')
↘ alpha	alpha parameter (function of 'x' and 'y')
↘ Phiq	Phiq parameter (function of 'x' and 'y')

Table 2: Input and Output of function

```
for i=1:nElem
    BDe=E(i,1);
    if BDe==nBoundary
        Ee=E(i,2:end);
        Xe=X(Ee,:);
        Ke=zeros(nen,nen);
        fe=zeros(nen,1);
        for j=1:nOfGauss
            N_j=N(j,:);
            Nxi_j=Nxi(j,:);
            w_j=wgp(j);
            xg=N_j*Xe(:,1);
            yg=N_j*Xe(:,2);
            J1D=Nxi_j*Xe;
            Ke=Ke+N_j'*N_j*alpha(xg,yg)*norm(J1D)*w_j;
            fe=fe-N_j'*(qn(xg,yg)-alpha(xg,yg)*Phiq(xg,yg))*norm(J1D)*w_j;
        end
        K(Ee,Ee)=K(Ee,Ee)+Ke;
        f(Ee)=f(Ee)+fe;
    end
end
```

Figure 3: SetNeumannBoundary.m code

3.2.3. SetDirichletBoundary

This function reduces the size of the matrix and global vector at the preset $\boldsymbol{\phi}$ values and replaces these values in the rest of equations.

SetDirichletBoundary.m	
↖ Kd	Global Matrix of K with Dirchlet cond.
↖ fd	Global Vector of f with Dirchlet cond.
↖ Nd	Nodes with Dirchlet cond.
↖ Nnd	Nodes without Dirchlet cond.
↖ Pd	Phi values in nodes Nd
↘ K	Global Matrix of K
↘ f	Global Vector of f
↘ Nd	Numero of boundary to set
↘ Pd	Phi value (function of 'x' and 'y')

Table 3: Input and Output of function

```
nNd=size(Nd);
nPt=size(K,1);
Ndd=[];
Pdd=[];
aux1=0;
aux2=0;
auxp=0;
for i=1:nNd
   aux1=Nd(i);
   auxp=Pd(i);
   for j=(i+1):nNd;
       if Nd(j)<aux1
          aux1=Nd(j);
          auxp=Pd(j);
       end
   end
   if not(aux1==aux2)
       Ndd=[Ndd; aux1];
       Pdd=[Pdd; auxp];
   end
   aux2=aux1;
end
Nd=Ndd;
Pd=Pdd;
Nnd=[1:nPt]';
Nnd(Nd)=[];
Kd=K(Nnd,Nnd);
fd=f(Nnd)-K(Nnd,Nd)*Pd;
```

Figure 4: SetDirichletBoundary.m code

4. Homework Meshes Solution

4.1.Solution

The solution of the meshes was made with ***HWsolver.m*** function.This function starts ***SolverDefault.m*** function for each of the homework meshes.

The SolverDefault.m function has preset the parameters of the homework, which are:

The domain parameters:

```
D='1';
Q='0';
```

The boundaries y = 0, y = 1 and obstacle as Neumann type with parameters:

```
qn='0';
alpha='0';
Phiq='0';
```

The boundaries x = 0 as Neumann type with parameters:

```
qn='1';
alpha='0';
Phiq='0';
```

The boundaries x = 1 as Neumann type with parameters:

```
qn='-1';
alpha='0';
Phiq='0';
```

Default the reference point is $\boldsymbol{\phi_{(0,0)} = 0}$.

The solver displays a report like the one in Figure 5 where:

$t1(s)$: Time needed to read the input.
$t2(s)$: Time to get boundary.
$t3(s)$: Time to assemble the global system.
$t4(s)$: Time for apply boundary conditions.
$t5(s)$: Time used to solve lineal system.
$t6(s)$: Time used to write solution.
$Phi(1,1)$: Phi value in (1, 1).

SOLVER HOMEWORK REPORT

#item	#mesh	#Elements	#Nodes	#N/elm	t1(s)	t2(s)	t3(s)	t4(s)	t5(s)	t6(s)	ttotal(s)	Phi(1,1)
1	1	458	1008	6	0.00955	0.67965	0.58015	1.41376	0.05988	0.09489	2.83788	1.33656626
2	2	889	1909	6	0.01457	2.38175	1.07591	2.69720	0.11663	0.15577	6.44184	1.33659650
3	3	2300	4816	6	0.03557	15.87867	2.75304	6.96673	0.77000	0.50829	26.91229	1.33660730
4	4	5136	10600	6	0.10736	108.76248	7.63991	19.73461	6.29833	1.11226	143.65496	1.33660953
5	5	9134	18708	6	0.19697	346.17642	13.61909	36.28275	27.89456	1.96559	426.13539	1.33661021
6	1	458	275	3	0.00484	0.28929	0.34394	0.07937	0.02175	0.04069	0.77987	1.32890204
7	2	889	510	3	0.00710	1.03690	0.66766	0.11305	0.05038	0.08880	1.96390	1.33237269
8	3	2300	1258	3	0.01665	6.43522	1.73181	0.19750	0.12539	0.17849	8.68506	1.33478303
9	4	5136	2732	3	0.03619	31.43363	3.82937	0.31005	0.40301	0.39148	36.40372	1.33572333
10	5	9134	4787	3	0.06395	98.49190	6.93708	0.50257	1.14946	0.68197	107.82694	1.33607291
11	1	246	838	8	0.00940	0.46034	0.55849	1.22115	0.03600	0.08789	2.37326	1.33658354
12	2	428	1416	8	0.01442	1.37724	0.94576	2.12644	0.07257	0.14506	4.68150	1.33660345
13	3	1161	3699	8	0.03677	9.97005	2.57422	5.81167	0.47061	0.37175	19.23507	1.33660956
14	4	2678	8362	8	0.08170	53.05696	5.95125	13.58826	3.26771	0.84318	76.78906	1.33661031
15	5	4606	14258	8	0.14175	155.36810	10.21332	23.72107	13.22618	1.43817	204.10857	1.33661047
16	1	246	296	4	0.00469	0.17356	0.24648	0.08600	0.01572	0.03635	0.56280	1.33239211
17	2	428	494	4	0.00643	0.47487	0.42617	0.11307	0.02779	0.05806	1.10640	1.33407543
18	3	1161	1269	4	0.01477	3.17050	1.15081	0.18986	0.08765	0.14853	4.76212	1.33565000
19	4	2678	2842	4	0.03229	15.59365	2.67022	0.35360	0.30455	0.33207	19.28639	1.33621090
20	5	4606	4826	4	0.05456	45.45209	4.62106	0.53627	0.97981	0.56589	52.20968	1.33635960

END SOLVER HOMEWORK

Figure 5: Solver homework report

Solution files can be found in the same location as the meshes, Solution.vtk, Which we can post-process with Paraview software. Reports are also saved in the same folder named report.txt.

Figure 6 shows a post processing by Paraview with the scalar variables ***Phi, pres*** (pres = -velo.velo / 2) and ***velo*** vector variable.

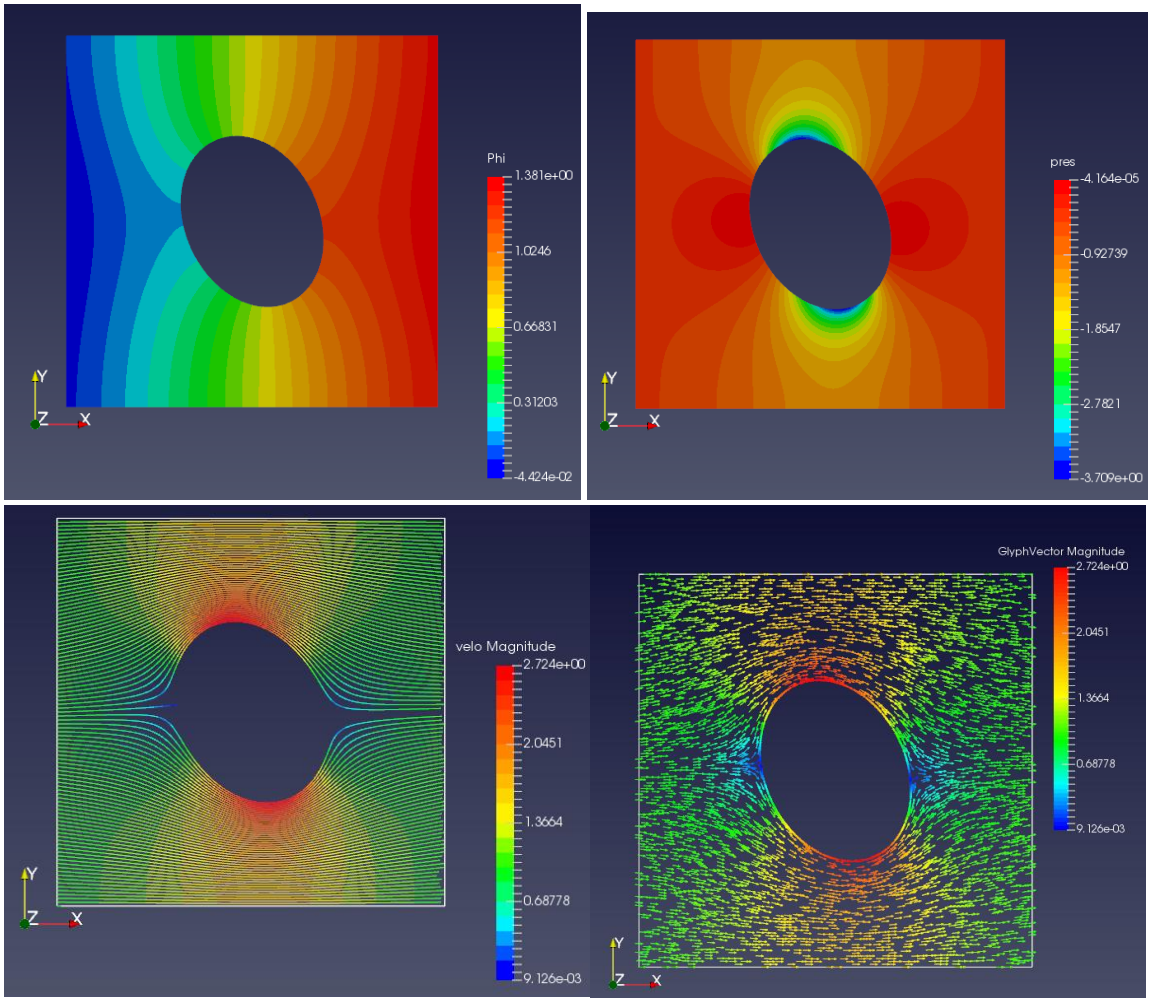

Figure 6: Post-processing with Paraview of PES_2D_quad_quad/mesh5.

4.2.Time solution

The times shown in the report, figure 5, are plotted by type of elements and are shown in figure 7.

The tendency of behavior of processing time with respect to the amount of nodes can be approximated according to Table 4.

The greater time difference can be observed for t4 which corresponds to the time of application of the boundary conditions, the trend shows more time for the second order elements.

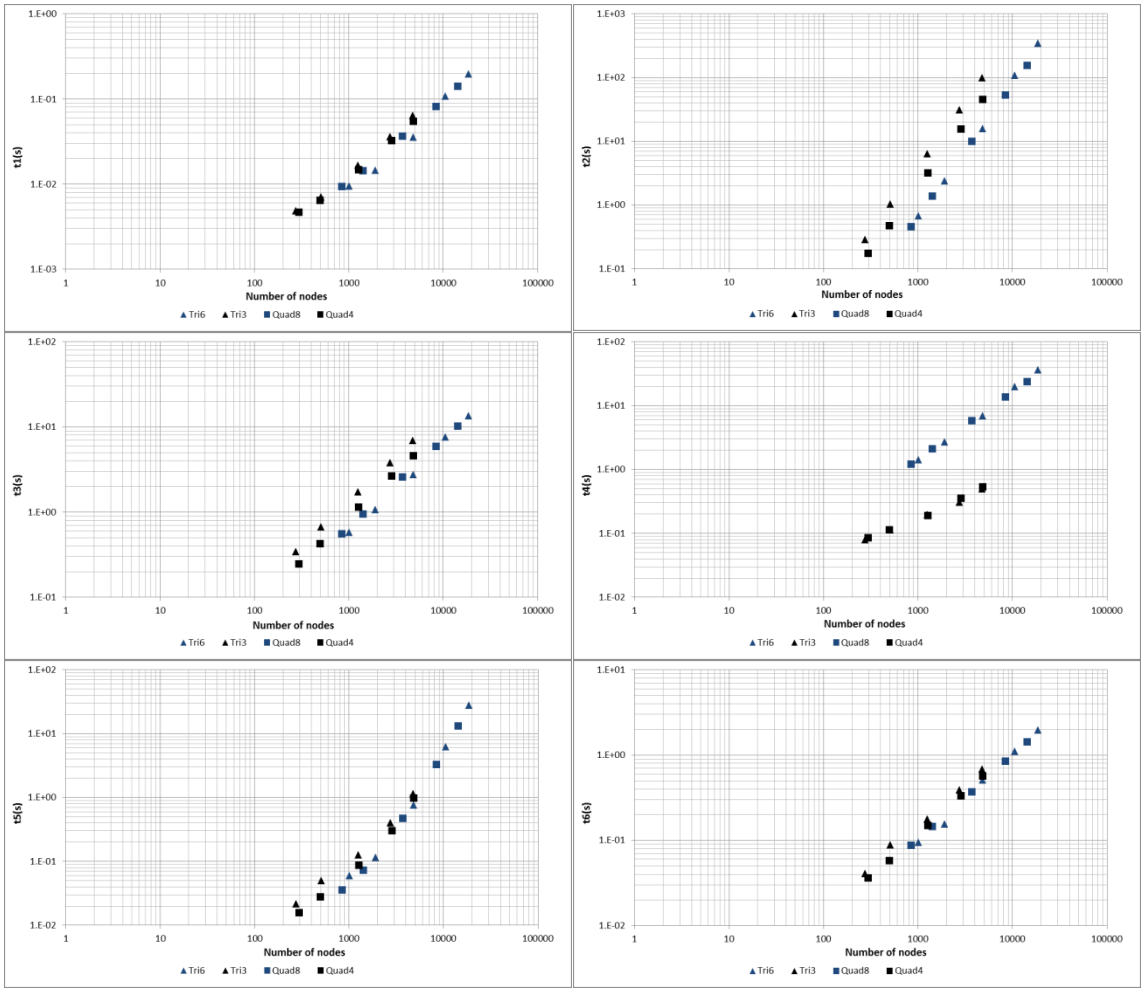

Figure 7: processing time by number of nodes.

Element type	t1		t2		t3		t4		t5		t6	
	≈ C.n^A		≈ C.n^A		≈ C.n^A		≈ C.n^A		≈ C.n^A		≈ C.n^A	
	C	A	C	A	C	A	C	A	C	A	C	A
Tri6	4.87E-06	1.07	2.27E-07	2.14	2.82E-04	1.10	5.76E-04	1.12	1.19E-08	2.17	6.39E-05	1.05
Tri3	2.14E-05	0.94	3.03E-06	2.04	1.03E-03	1.04	1.82E-03	0.67	1.11E-05	1.34	1.54E-04	0.99
Quad8	1.07E-05	0.99	4.07E-07	2.07	5.27E-04	1.04	9.60E-04	1.06	2.31E-08	2.10	9.41E-05	1.01
Quad4	4.28E-05	0.84	1.97E-06	2.00	6.73E-04	1.05	2.64E-03	0.61	4.58E-06	1.42	1.26E-04	0.99

Table 4: Trend of processing time behavior.

4.3.Error of solution

Taken as the reference solution, the mesh 5 of the elements quad8 can estimate the relative error by comparing the $\boldsymbol{\phi}_{(1,1)}$ values. The behavior of this error with respect to the number of nodes can be observed in *figure 8* and *table 5*.

The fall of the error in the elements type Quad8 is much faster than the rest of elements followed by Tri6.

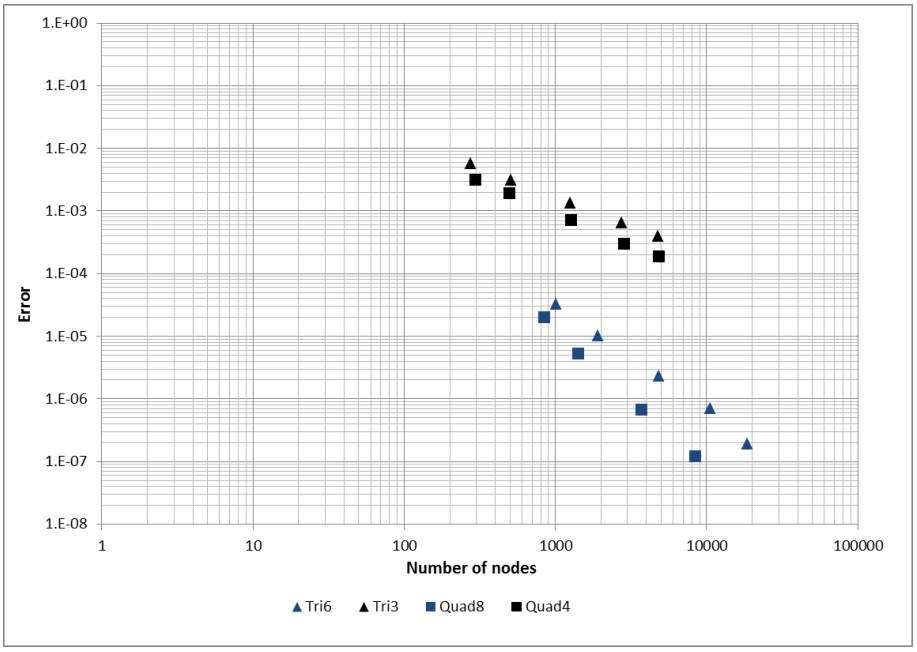

Figure 8: Relative error by number of nodes.

error		
Element type	≈ C.nA	
	C	A
Tri6	4.53	-1.71
Tri3	1.07	-0.93
Quad8	52.46	-2.21
Quad4	1.07	-1.02

Table 5: Error behavior.