

UPC

Solution of the Poisson problem

Programming for Engineers and Scientists

Sònia Garrido Ballart

Alba Navarro Casanova

0. Statement of the problem

In the initial classes of PES we extended an existing Matlab code implementing FE for a Poisson equation,

$$\begin{cases} \nabla \cdot (\nu \nabla u) = s & \text{in } \Omega, \\ u = 1 & \text{on } \Gamma_{Inlet}, \\ u = 0 & \text{on } \Gamma_{Outlet}, \\ (\nu \nabla u) \cdot \mathbf{n} = 0 & \text{otherwise} \end{cases}$$

where the source term $s = 0$ and the diffusivity $\nu = 1$ are given.

This assignment consists in further extending that code. We ask you to develop one single Matlab code able to solve the Poisson problem, accounting for the following options and features¹:

- Read a general mesh and boundary condition from input files.
- Solve 2D/3D and steady state problems
- Use triangular/tetrahedral, quad/hexahedral using linear or quadratic elements.

0. Introduction to the code

The code of the problem can be download from the following repository of GitHub https://github.com/soniagarrido/PES_HW1_GarridoSonia_NavarroAlba

We can solve this problem with eight different types of meshes, we have also the option for solving our Poisson's problem in two dimensions or in three. The idea is that when we initialize the program we can chose how we want to solve the problem. The election of the type of problem is made when running the Main of the Matlab code. When we execute the problem the eight options that can be solved are:

Options for solving the problem in two dimensions (2D)

1. Quadrilateral elements with linear shape functions: 2D_quad_linear
2. Quadrilateral elements with quadratic shape functions: 2D_quad_quad
3. Triangular elements with linear shape functions: 2D_tri_linear
4. Triangular elements with quadratic shape functions: 2D_tri_quad

Options for solving the problem in three dimensions (3D)

5. Cubic elements with linear shape functions: 3D_quad_linear
6. Cubic elements with quadratic shape functions: 3D_quad_quad
7. Tetrahedral elements with linear shape functions: 3D_tri_linear

8. Tetrahedral elements with quadratic shape functions: 3D_tri_quad

We can also enter the value of diffusion that we want.

Once we have chosen the type of problem that we want to solve the Main code will read the node and element files corresponding to each problem to create the mesh and the matrix that we need to solve the linear system. The code also solves the linear system of equations to do this we have specified one function for every case where the shape functions its derivatives and the integration points and weights are defined. We also need to impose the boundary conditions and they are specified in every case using the group field that we have to solve the problem.

Finally we have a postprocess part for every case that allows us to see the results using ParaView and we can observe the mesh and the results that we obtained in the different cases of study.

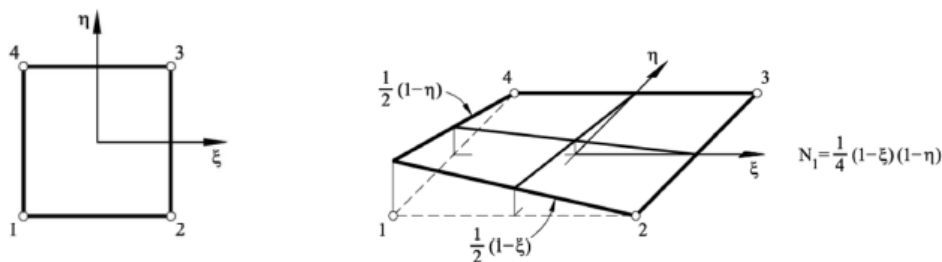
1. Problem in two dimensions (2D)

Solving the problem in two dimensions corresponds solving the cases 1 to 4. In these different cases we use quadrilaterals or triangular elements depending on the case and linear or quadratic shape functions.

Now we are going to explain some comparisons with the different cases and how was their accuracy, running time, etc

1.1. Quadrilateral elements with linear shape functions

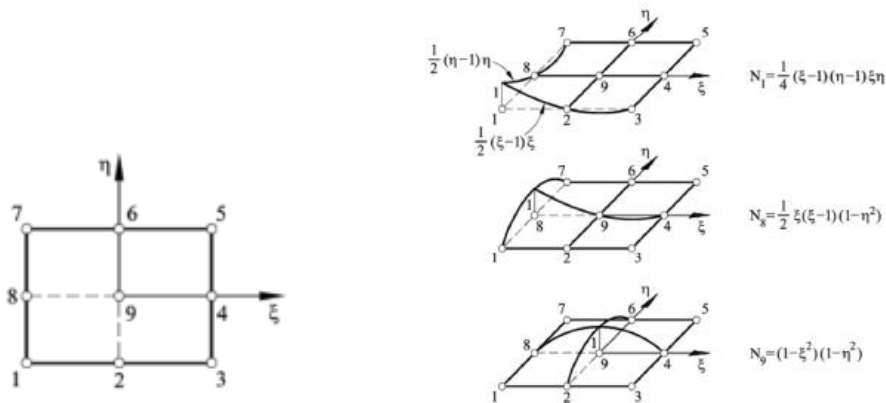
As we know the number of element nodes in this case is 4 and we have elements like these:



To solve this type of problem we have created a function called C2D4 where the shape functions and the derivative shape functions are defined. We need them to compute the stiffness matrix and get the solution of the problem.

1.2. Quadrilateral elements with quadratic shape functions

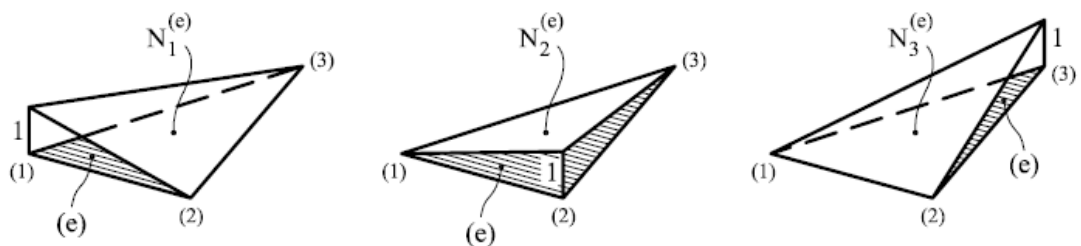
In a quadratic element with quadratic shape functions we have 8 element nodes like we can see in the figure:



We have quadratic shape functions and also the gauss points for this case are shown in the file C2D8.m file.

1.3. Triangular elements with linear shape functions

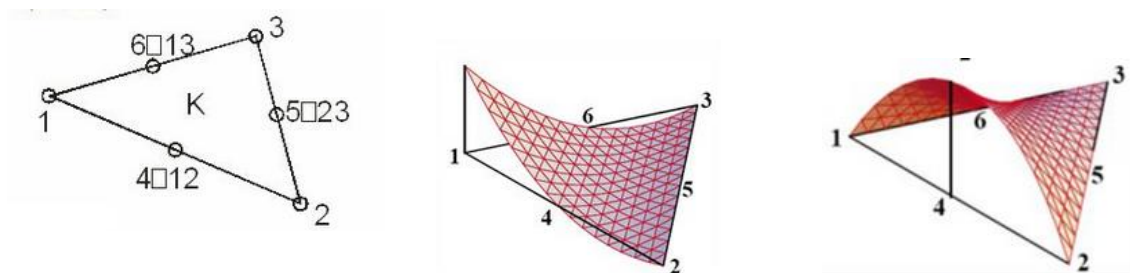
In this case we have a mesh formed by elements like are shown in the figure:



As we can see in the Poisson's problem for this case the number of element nodes is 3 and the shape functions are linear. The gauss points of this case are shown in the file C2D3.m

1.4. Triangular elements with quadratic shape functions

We have a mesh formed by elements like are shown in these images:



In this case the number of elements nodes is 6, and here we have quadratic shape functions. The gauss points for this case are shown in the file C2D6.

2. Problem in three dimensions (3D)

The problems that we have to solve in 3D are the same that in the previous case. The main difference is that the elements that we use are tetrahedral instead of triangular elements and cubic elements instead of quadrilateral elements.

Solving this problem corresponds to solve case 5, 6, 7 and 8 shown below :

5: 3D_quad_linear;

6: 3D_quad_quad

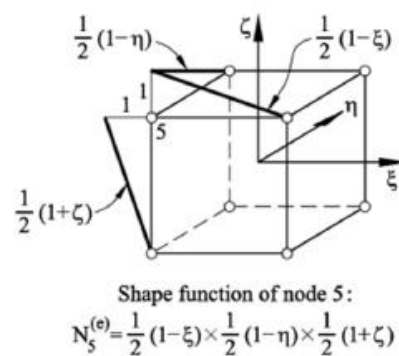
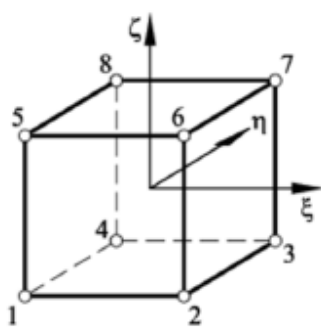
7: 3D_tri_linear

8: 3D_tri_quad

As we are working in three dimensions our elements have more nodes that in the 2D case.

2.1. Cubic elements with linear shape functions

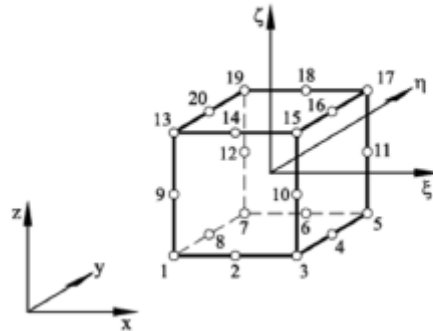
In a similar way as in the tetrahedral case the linear cubic elements have eight nodes and we have to find shape functions that guarantee the value one on the node that we are studying an zero on the other ones.



The shape functions and the derivatives of the shape functions are defined in the function Matlab file called C3D8.

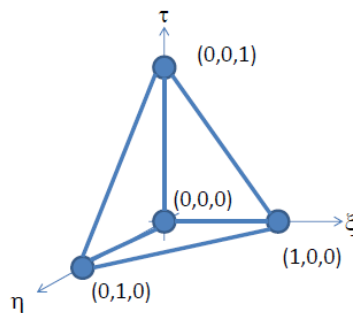
2.2. Cubic elements with quadratic shape functions

The quadratic cubic elements have 20 nodes and the shape functions and derivative shape functions for this type of elements are defined in the C3D20 function of the matlab code.



2.3. Tetrahedral elements with linear shape functions

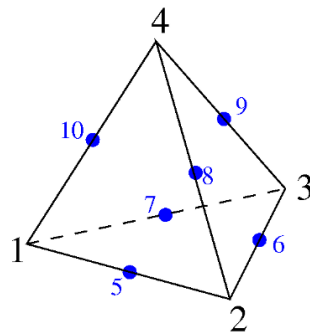
The number of nodes for this type of elements is 4 nodes for element, as we can see in the following figure.



As we have linear shape functions we do not need to add extra point to solve our problem. To solve this type of problem we have created a function called C3D4 where the shape functions and the derivative shape functions are defined. We need them to compute the stiffness matrix and get the solution of the problem.

2.3. Tetrahedral elements with quadratic shape functions

For this second type of element we add one extra point for edge so in the end we have a tetrahedral with 10 nodes as we can see in the following figure.



We have more shape functions and derivative shape functions as we increase the number of nodes on the elements. This shape functions and the derivatives of the shape functions are defined in a function called C3D10.

3. Comparisons of the solutions

In the next tables we can see how change the running time if we use one method or other. Also it is shown the stiffness matrix dimensions for the different meshes.

3.1 Problems in 2D

Element	Shape function	Running time (s)	Dimensions of K
Quadrilateral	linear	2,27408	258x258
	quadratic	2.615.557	732x732
Triangular	linear	2.900.859	243x243
	quadratic	3.237.881	889x889

We can see that the mesh that has less running time is the one that uses quadrilateral elements with linear shape functions. And the mesh that takes more running time is the triangular elements with quadratic shape functions, we can see also that the matrix K in this one is more larger than the others.

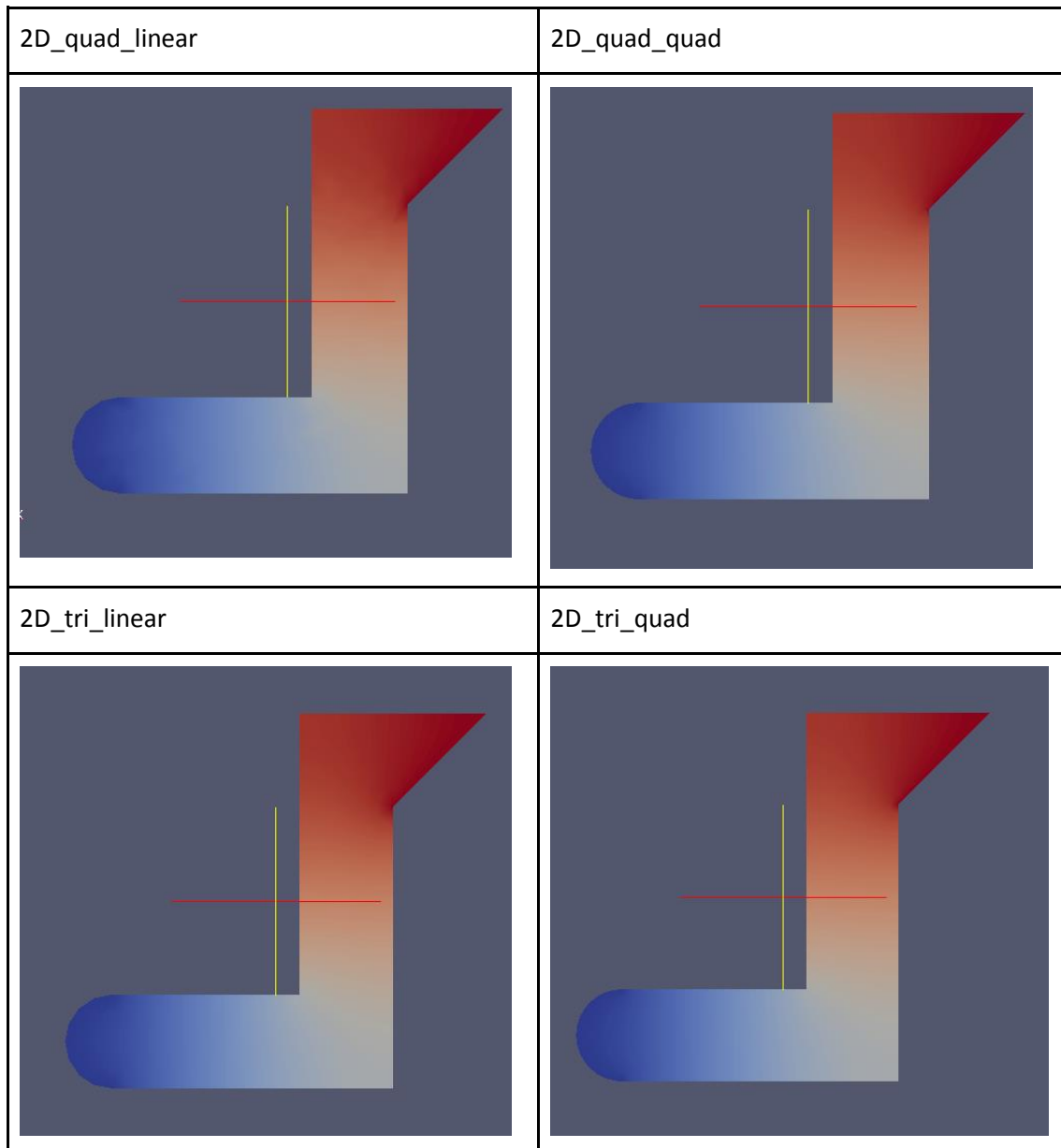
3.2 Problems in 3D

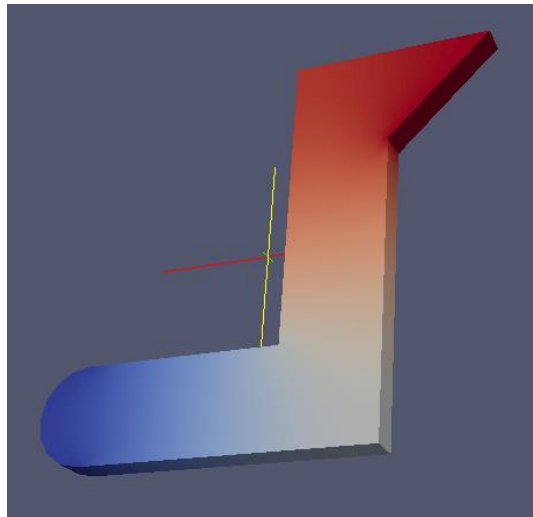
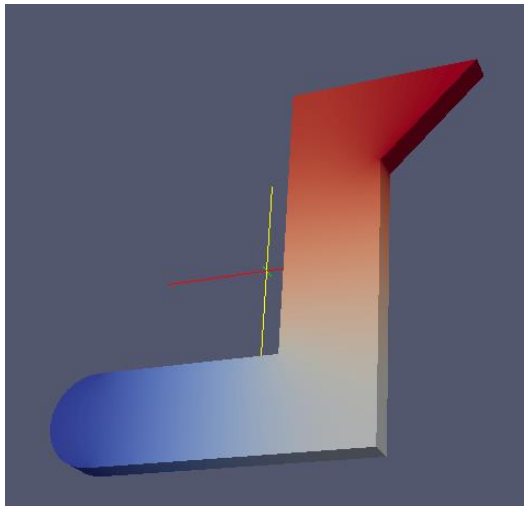
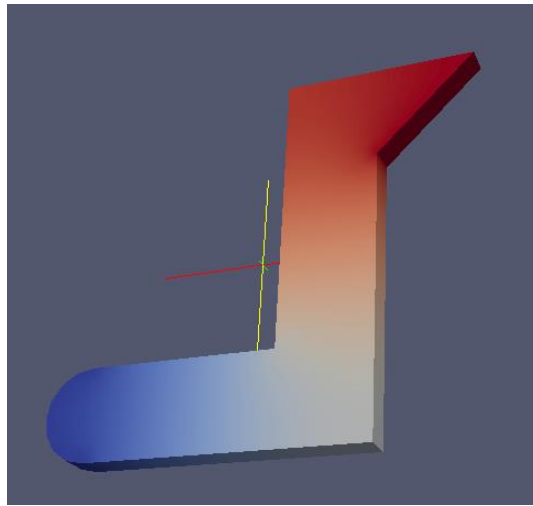
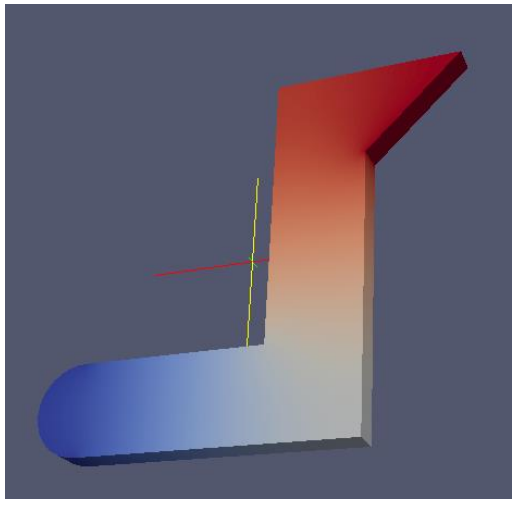
Element	Shape function	Running time (s)	Dimensions of K
Cubic	linear	2,556663	777x777
	quadratic	5,339130	2723x2723
Tetrahedral	linear	2,336580	738x738
	quadratic	8,863932	4526x4526

In three dimensions we can see that the mesh that has less running time is the one that uses tetrahedral elements with linear shape functions. And the mesh that takes more running time is the tetrahedral elements with quadratic shape functions, and as it happens in 2D the stiffness matrix in this one is more larger than in the other cases.

4. Solutions in ParaView

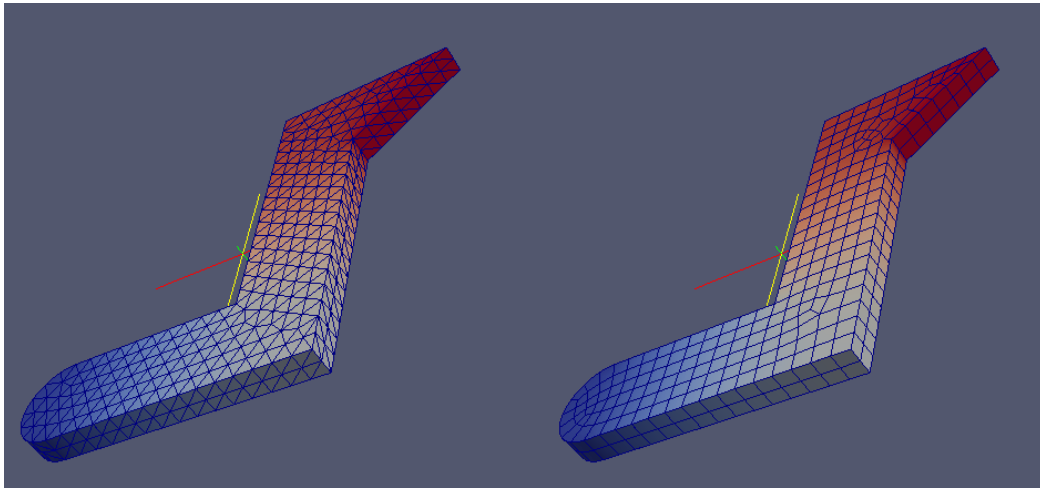
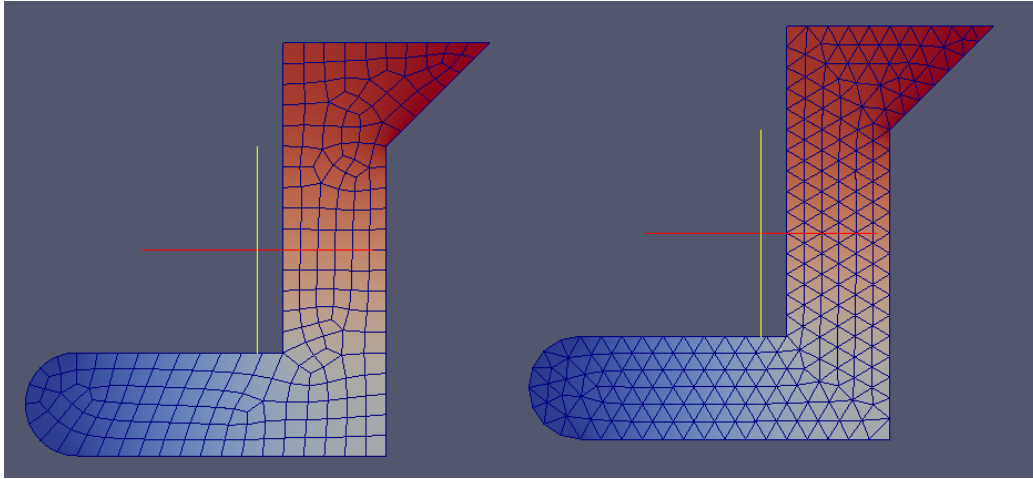
After running the code for the eight cases we can see the solutions in ParaView. As we are solving the same problem we get the same solution, the only difference is the type of elements and the numbers of nodes that have the figure.



3D_quad_linear	3D_quad_quad
	
3D_tri_linear	3D_tri_quad
	

As we can see in the figures that we obtained in paraview we get the same result for all the different meshes so the best type is which has less running time because its computational cost is cheaper. In this case the best option is use for 2D quadrilateral elements with linear shape functions and for 3D tetrahedral elements with linear shape functions.

In the following figures we can see how the triangular, quadrilateral, tetrahedral and cubic meshes look like. In the mesh we can not appreciate if we are using linear or quadratic shape functions.



In general when we are creating a mesh to compute the solution in a structure or in a domain we prefer using in 2D quadrilaterals over triangles and in 3D bricks over wedges, or wedges over tetrahedral.