Master's Degree Numerical Methods in Engineering

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Programming for Engineering and Scientists

Homework 2.b: Design of a FE code in C++

Author:
Luis Ángel Avilés Murcia
luis.angel.aviles@upc.edu
Mariano Tomás Fernandez
m.tomasfernandez@gmail.com
Shardool Kulkarni
shardoolkulkarni1996@gmail.com

Professor:
Sergio Zlotnik

June 15, 2020
Academic Year 2019-2020

Contents

1 Introduction

In this assignment we develop a Finite Element Method code in C++ to solve a 2-dimension Poisson problem, with the conditions shown in Equation (1).

$$\begin{cases} \Delta u = 0, & \text{in } \Omega \\ \boldsymbol{\nabla} \cdot u \cdot n = -1, & \text{on } \Gamma_{in} = 0x(0,1) \\ \boldsymbol{\nabla} \cdot u \cdot n = 1, & \text{on } \Gamma_{out} = 1x(0,1) \\ \boldsymbol{\nabla} \cdot u \cdot n = 0, & \text{on } \partial\Gamma \backslash \Gamma_{in} \cup \Gamma_{out} \\ u(0,0) = 0 \end{cases} \tag{1}$$

where Ω is the computational domain, $\partial\Omega$ its boundary, and $\mathbf{n}$ is the outward normal vector.

The velocity field is obtained from potential such as

$$v_x = \frac{\partial u}{\partial x} \qquad\qquad v_y = \frac{\partial u}{\partial y} \tag{2}$$

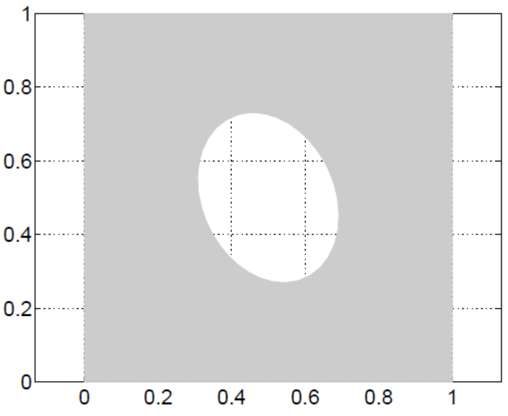

Figure 1: Domain Geometry

The idea of this assignment is to follow the structure depicted in the previous assignment (2.a), mention the changes introduced, and explain the reason for each of these.

2 Design of the Code

2.1 Changes to the presented design

2.1.1 Classes

From the different classes mentioned in the design of the code, only `quadrature`, `element stiffness` and `shape function` class were implemented. The `fem solver` and the `global matrix` classes were condensed into the main loop. Each of the classes used both Public and Private variables, class `element stiffness` has a constructor to initialise the variables that need the class.

2.1.2 Structures

Datatype of structures were used to perform certain operations. `Constraint` structure classifies the Neumann boundary conditions on the problem and `element` is a structure for accessing the connectivity of individual elements. Apart from these, specialised functions are implemented which perform specific tasks, such as applying the boundary conditions and calculation of the stiffness matrix among others.

2.2 External Libraries

`Eigen` was used to define vectors and matrices, and perform all the linear algebra operations extensively throughout the code. We use the `<isstream.h>` which is a header providing the standard input and combined input/output stream classes. Within this, we use the `ifstream` class to read the mesh data files.

2.3 Selection of the coding software

We preferred to use the community version of Visual Studio Code instead of the Qt Creator for student, given the first is a light program code oriented program with several debuggers and tons of information available on the internet. The visual studio project interface makes it much easier to track all the header files, external dependencies and source files to name same. Visual Studio also offers easy integration with `Eigen` on Windows. All the code for the project was done in a single file, leading to an extensive code.

2.4 Analysed cases

Due to lack of time, the code can solve the problem only for linear triangular elements. Other mesh conditions can be extended adding conditionals to the element shape (triangular or quadrilateral) and its degree of interpolation (linear and quadratic elements). Extending the finite element code from MATLAB to C++ was not an easy task and several differences came into light while doing it. The problems we experienced throughout the code took us valuable time which avoid us from implementing more general mesh conditions.

2.5 Things to improve in our code

We recognize the limitations of our code and for that reason, the following improvements can be implemented:

- Extend the code to quadrilateral elements
- Add quadratic element interpolation
- Remove the lines to compute the stiffness matrix from the main code and become a class with two methods, `element.stiffness` and `element.force` per element.
- Implement a class to read the external files of nodes and connectivities.
- Code a class to extract the boundaries from the meshes.
- Create a class to assembly the global stiffness matrix.
- Implement the class solver, to take all the matrices and vector and solve the system.

2.6 How the code works

The code is Visual Studio project, so it needs to be open as a project in Visual Studio using the `.sln` file. It is necessary to add the Eigen libraries to the project directory. Inside the code the route to the meshes are local, so surely it is necessary to change the path to the place where the Meshes folder is located. If is the first time opening VS it is necessary to build the project in the Build menu, then the program can be run with the start button.

3 Results

3.1 Results triangular mesh

As mentioned before the code runs only for linear triangular elements, therefore these results are reported. From the results, we observe general agreement for different mesh sizes from coarse to fine. The results for the coarsest and the finest mesh are presented below in Figure 2 and Figure 5. In Figure 4 the streamlines are merged along with the results.

Given the fact that the obtained results are saved into a *.vtk* file, as we have done for the MATLAB part and the incapacity of the software to plot the results, the PARAVIEW software was used to plot them. The results show an expected result of potential flow around a cylinder, the streamlines show symmetry around the cavity. We can also see the stagnation point at around $y = 0.5$ both in Figure 4 and Figure 5.

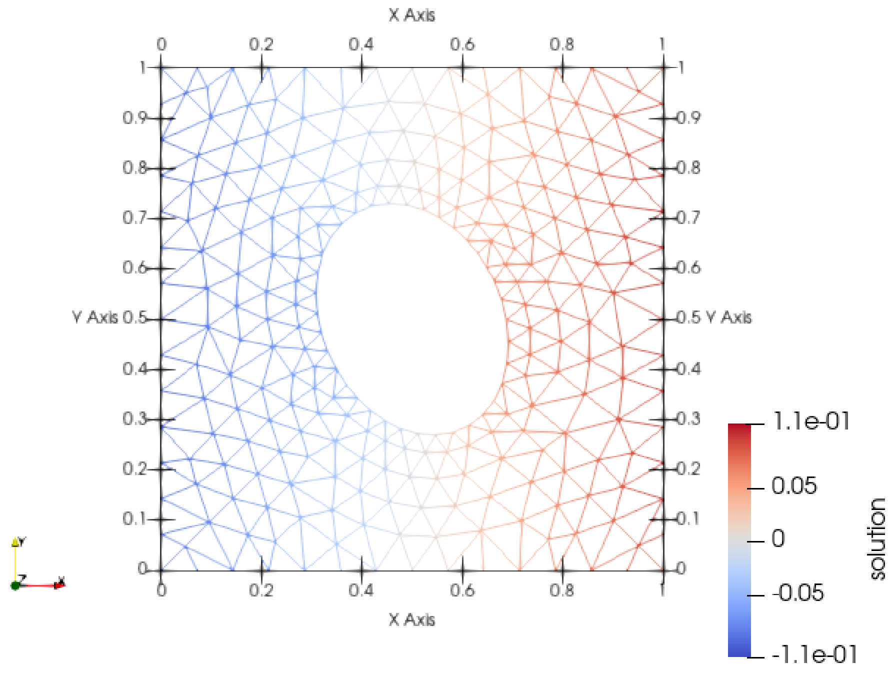

Figure 2: Solution u for coarsest Mesh: 275 nodes

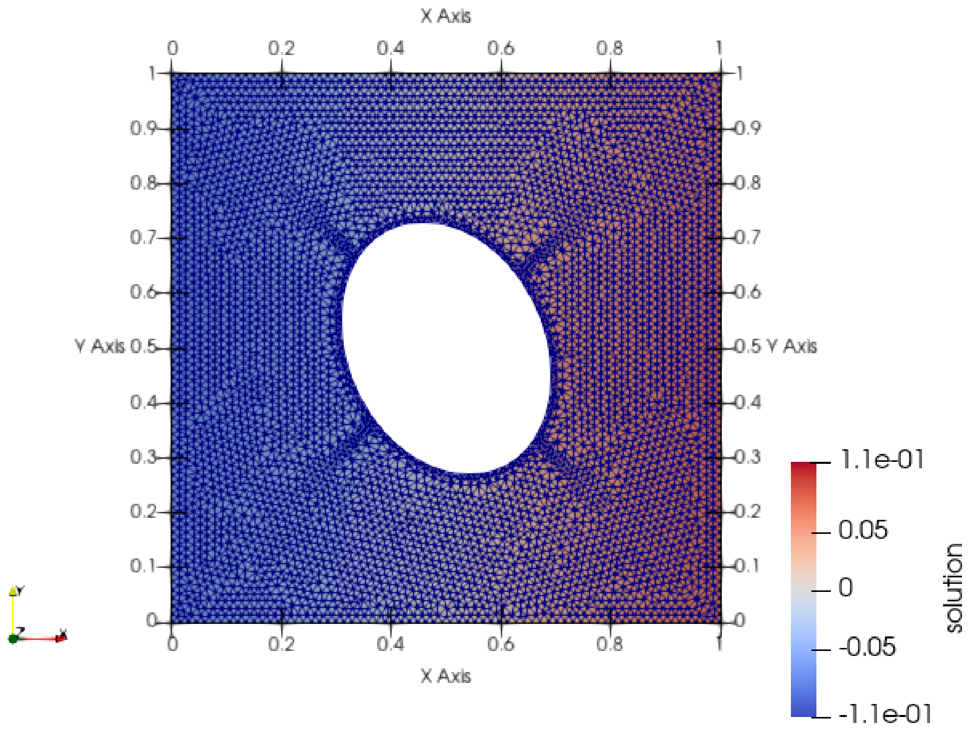

Figure 3: Solution u for finest mesh: 4787 nodes

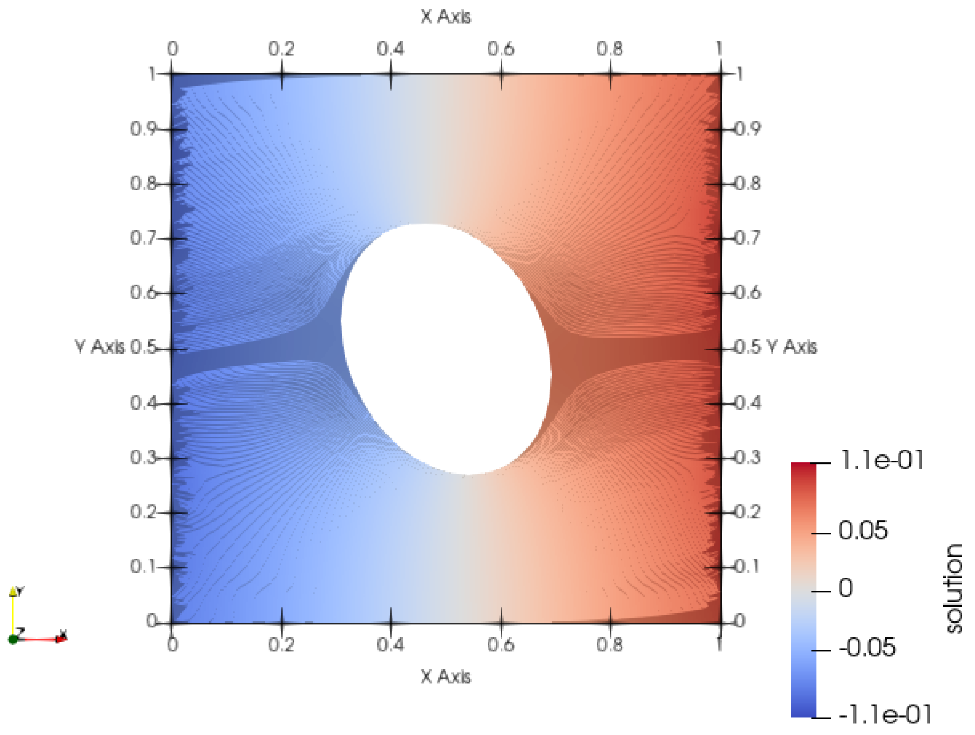

Figure 4: The streamlines for Coarse mesh

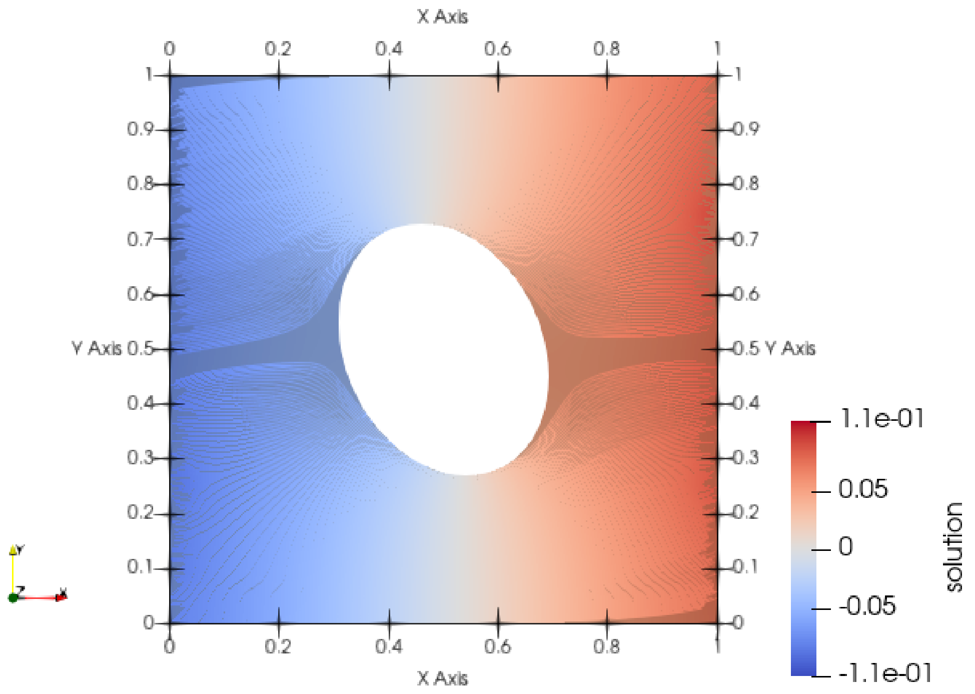

Figure 5: Streamlines for fine mesh

3.2 Convergence

The convergence analysis was performed considering the finest mesh as the '*exact*' solution, we look at the L_∞ norm, the difference in absolute values of the maximum solution values, we observe that the error decreases as we increase the number of elements. A linear decrease is expected as we are dealing with linear triangular elements, and the trend line in Figure 6 shows an R^2 of about 95% which matches the expectations.

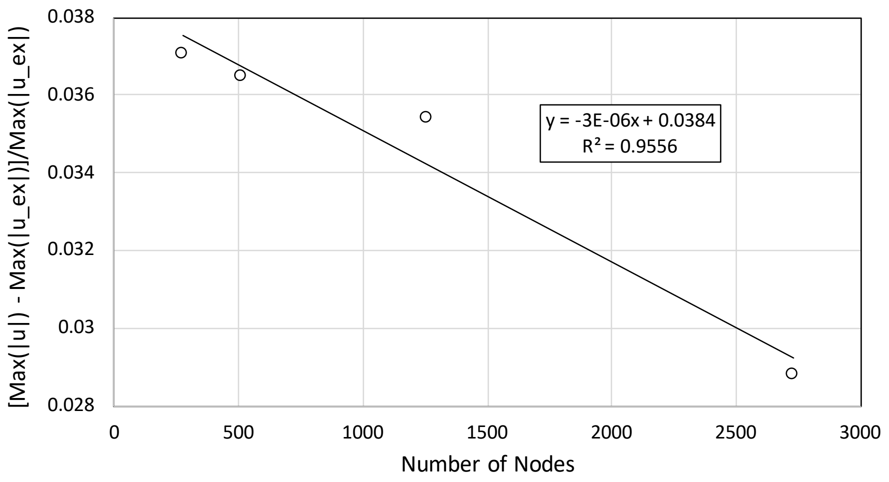

Figure 6: Converging Error for linear Triangular elements

4 Conclusions

In this assignment a **C++** Finite Element Method code was developed in order to analyse a Laplacian problem in two-dimensions. Even though some difficulties were encountered when implementing different element shapes and order of interpolation, the implemented version of the *linear triangular elements* shown credible results.

Moreover, when analysing the convergence of these results it is seen that the L_∞ norm of the error is reduced for finer meshes as expected. In this way, the code is said to be limited to a certain type of element but properly working.

Further improvements can be implemented to work with liner and quadratic quadrilateral and triangular quadratic elements.