Homework 1. Part a.

Programming in Science and Engineering

Nino Guzmán: *ninogdo@ciencias.unam.mx*

David Encalada: *daviden007@hotmail.com*

Manousos Galanakis: *manousosgalanakis@gmail.com*

Shridharan Suresh: *shridharan25@gmail.com*

March 2020

1 Introduction

First, we give a quite brief introduction to what the Finite Element Method (FEM) consists in. The FEM is a numerical method to approximate a solution using a discrete problem instead of the continuous one. It has been used on different fields and disciplines [1]. Here we explain the general idea of the method and the process to follow for their implementation.

We begin with a partial differential equation restricted to a boundary value problem. The first step will be to obtain the weak form of the problem, typically through the weighted residuals method, and then find a new problem to solve. Afterwards, we discretise the weak form, choosing the proper test functions and the desired shape functions. This will give us a system of equations to be solved. If the initial PDE problem is linear then we will get a liner system of equations. Then, as a final step, we must analyse our solution in order to obtain convergence, consistency and accuracy of the implemented method and obtain the final solution. All of this done in the postprocess stage, as we will mention later on. We can see a scheme of this procedure in Fig. (1).

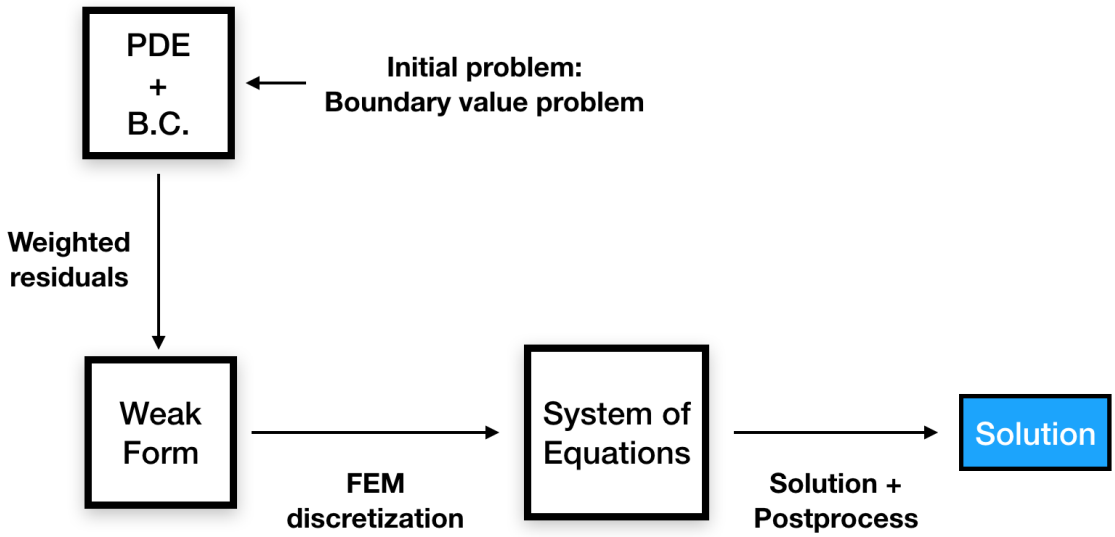

Figure 1: Scheme of FEM procedure.

The goal of this report is to give the complete structure for implementing a MATLAB code that uses the Finite Element Method to solve partial differential equations.
First of all, we want to distinguish three main parts of the code: **preprocess**, **process** and **postprocess**.

- **PREPROCESS**
  In this step we create the necessary elements for the code in order to solve it numerically. In here we create two main functions: **referenceElement()** and **createMesh()** and one more that will have the physical information **materialProperties()**

- **PROCESS**
  In this second part, we create the system of equations and it is solved after imposing boundary conditions. We must say that depending on the type of boundary conditions we can impose it directly (no additional function) or through out some new function. The more general case would be create a function to impose them, here we present two functions for each case of boundary condition: **dirichletBC()** and **neumannBC()**. Specifically we create the matrix **K** and the RHS vector $\vec{f}$. All this through out the function **computeSystem()**.

- **POSTPROCESS**
  In the third step, we analyse our solution either numerically and physically. The convergence of the implemented method is obtained using a new function **computeError()** and finally plot the solution using a function **plotSolution()** with regard to interpret the results.

The whole process and the functions involved in each step are shown in Fig. (2) and each one is explained in the following sections.

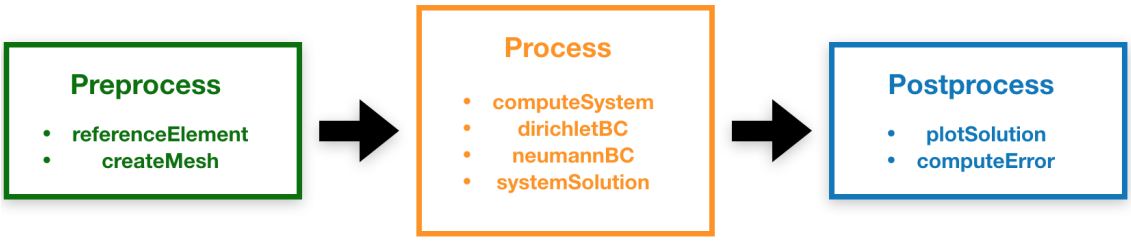

Figure 2: FEM procedure divided in preprocess, process and postprocess.

2 List of Data Structures

A data structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way. We select these data structures based on which type of operation is required. There are some built-in data structures such as integer, character etc, called as Primitive Data Structures. And there are other complex data structures such as array, list etc, that are user-defined to store large data.

As for FE code design we define multiple variables that are of the same data type we have listed below the different data structures that are needed:

- **Integer**
- **Character**
- **Array**
- **List**
- **Float**
- **Mesh**

3 FEM implementation process

3.1 Preprocess

In this section, we show the functions that are implemented only in the preprocess but that can be used in several parts of the code.

referenceElement()	
Inputs	• Dimension • Type of element • Degree of interpolation
Outputs	• Shape functions • Gauss points coordinates and values • Weights of corresponding Gauss points • Nodes coordinates of shape functions • Jacobian (derivatives of shape functions)
Description	Implements the reference element in 1D, 2D or 3D, depending on the degree of interpolation and the type of element.
Comments	Depending on the dimension we will have different type of arrays as the output.

createMesh()	
Inputs	• Dimension • Domain coordinates • Number of elements (*) • Type of element
Outputs	• Nodes coordinates • Connectivity matrix
Description	Implements the mesh in 1D, 2D or 3D, using different types of elements.
Comments	(*) Depending on the dimension, we must define number of nodes in each degree of freedom. In here we could also define the length between each node.

materialProperties()	
Inputs	• Mesh (nodes coordinates)
Outputs	• Prescribed values on the desired nodes
Description	It fixed the values of the material properties that could depend on space.
Comments	The type of output array will depend on the dimension and the material properties.

3.2 Process

In this section is where the main computations are made. We construct the linear system (or nonlinear if it is the case) and we solve it numerically using some specific quadrature. We list below the functions used in this part:

computeSystem()	
Inputs	• Mesh (specifically the nodes coordinates $\vec{X}$ and the connectivity matrix $\mathbf{K}$ • Reference element outputs: IP coordinates, IP weights, Jacobian, number of nodes, number of elements. • Source term (*) • Material properties
Outputs	• Global matrix $\mathbf{K}$ • RHS vector $\vec{f}$
Description	It creates the system to be solved taking into account material properties.
Comments	We consider this function is the core of the FEM. The quadrature used to approximate the integrals can be implemented directly inside this part of the code or through some new function. The boundary conditions are imposed after creating the system of equations. (*) The source term can be computed directly here or using another function, depending on the complexity. It is common to be constant or zero.

dirichletBC()	
Inputs	• Nodes coordinates
Outputs	• Nodes over the boundary • Nodal values over the boundary nodes
Description	This functions find the nodes over the boundary in 1D, 2D or 3D and it fixes the known values over these nodes.
Comments	Depending on the complexity of the mesh and the dimension this function could be eliminated and instead imposing the DBC directly.

neumannBC()	
Inputs	• Nodes coordinates • Material properties • Reference element outputs: IP coordinates, IP weights, Jacobian, number of nodes, number of elements.
Outputs	• Nodes over the boundary • Nodal values over the boundary nodes (*)
Description	This function computes a contour integral where Neumann boundary conditions are imposed.
Comments	To compute the Neumann boundary conditions we must use a quadrature to solve the integral, as we mentioned before we could solve directly or use another function to call a specific quadrature. Once the integral is computed over the boundary it will contribute to the RHS vector $\vec{f}$. (*) In this case these values are the ones found solving the integral.

systemSolution()	
Inputs	• Matrix $\mathbf{K}$ and RHS vector $\vec{f}$ • Nodes over the boundary • Nodal values over the boundary nodes
Outputs	• Nodal values of the solution $\mathbf{u}$
Description	This function first imposes boundary conditions and then solves the system and computes the solution.
Comments	The necessary modifications to $\mathbf{K}$ and $\vec{f}$ to impose BC are made inside this function. Moreover, it will depend on the complexity of the problem if we create this function or solve the system directly.

3.3 Postprocess

The postprocess' goal is to analyse the solution we found in the previous stage. This includes plot the solution and interpret the results to see if it makes sense. On the other hand, we also must analyse how good the implemented method was, i.e., we must check the convergence of the method. The functions used in this part are:

plotSolution()	
Inputs	• Mesh • Solution **u**
Outputs	• plot of the solution over the domain
Description	Plots the solution over the domain.
Comments	It will be helpful to plot different solutions for several values of the specific parameters or material properties in order to analyse the results.

computeError()	
Inputs	• Solution **u** • Mesh
Outputs	• Total error
Description	This function computes the error of the approximated solution either comparing with the analytic solution or approximating it somehow. It is done element by element.
Comments	We could use this function to compute the $\mathcal{L}^2$ and the $\mathcal{H}^1$ norms and analyse if the method is behaving as expected.

4 Dependence graph

The scheme of the dependence of the functions is shown in the Fig. (3). First, the grid is defined with the function **createMesh()** and inside the function **referenceElement()** are all the properties to solve numerically the integrals that will leave us to the system of equations. The matrix **K** and the RHS vector $\vec{f}$ is computed using **computeSystem()**. The corresponding boundary conditions are computed in **neumannBC()** and **dirichletBC()**. The system is solved by **systemSolution()**, the function modifies **K** and $\vec{f}$ according to the BC. Finally, the results can

be analysed using **plotSolution()** and **computeError()** to see if the method is implemented correctly and make sense physically speaking.

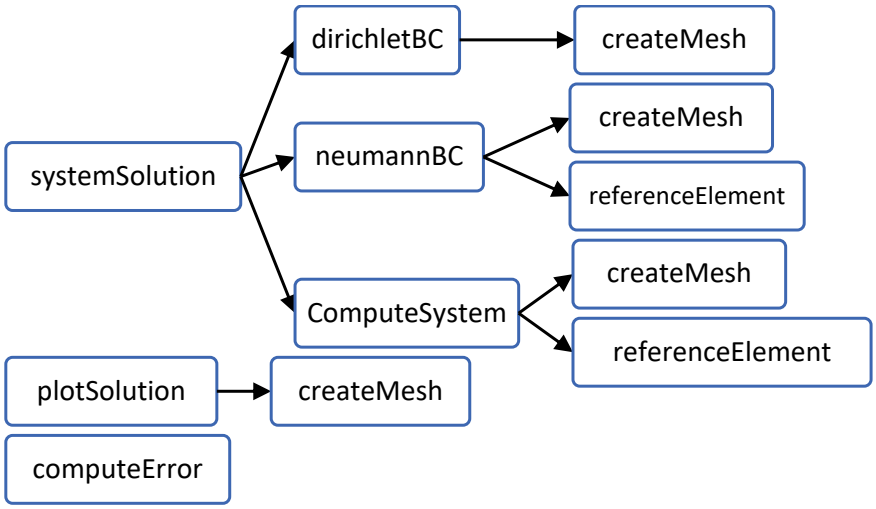

Figure 3: Scheme of function dependence.

Other alternative follows a similar scheme. The difference is to create a function that assigns the physical properties, that might depend on space, to the elements to compute matrix K. In the alternative scheme the **dirichletBC()** function calls **computeSystem()** and **neumannBC()**.

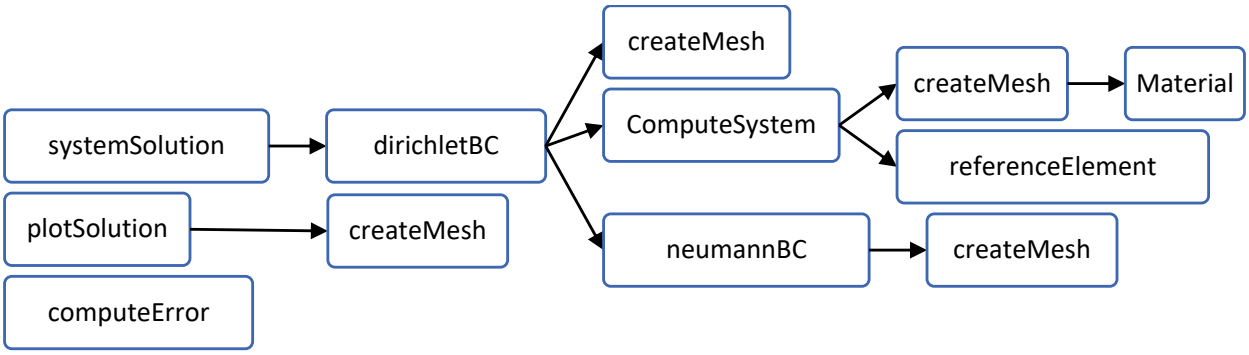

Figure 4: Alternative scheme.

5 Conclusions

We decided to divide the process in three stages in order to clarify how to implement the code. Although this help us to define the functions and structures needed for the implementation, we still need to improve some points in our design.

Some advantages of this design are the following: first of all, this design splits the procedure in three main stages which makes easier to define the tasks inside each one. The design is general. It is supposed to work for several dimensions and different physical problems taking into account several parameters and material properties. Moreover, this design includes an analysis either the physical aspects and the numerical ones which would be useful to improve if needed.

Although we consider our design is quite general and robust, we still have to list some specifications that can be done. The first thing to point out, is that we lack a function that could compute several quadratures for 1D, 2D or 3D, besides we could also improve how the BC are imposed, perhaps creating a new function. Finally, we could also create a set of functions that can solve nonlinear system of equations, because this design is constructed mainly for linear problems.

References

[1] O.C. Zienkiewicz, R.L. Taylor & J.Z. Zhu , *The Finite Element Method: Its Basis and Fundamentals*, Sixth edition (2005)