PES - Homework 1

Students:
Lei Pan
Marcello Rubino
Enrico Marin

1) INPUT DATA (in this section we describe the tables containing the input data)

-The first input data that need to be used are about the geometrical and material informations. In order to do this the first element is the matrix ***nodalCoordinates***. Inside, for each row, which defines each node, coordinates X and Y are written.

-Now we can focus on each element. Inside the table ***elementInfos***, for each row that describes the Id of the element, there are the informations about the connectivity of the nodes. There are 10 columns: the first 4 of them are about the Id of the master nodes that shows the nodes of the element, written in counterclockwise order. If the element is triangular, the fourth row contains 0, otherwise, obviously, contains the fourth node's Id. If the element is linear, the column 5-6-7-8-9 are 0 (it means that there are no extra nodes inside the faces of the elements); otherwise, in case of quadratic element, we need to fill these columns as well (still following the counterclockwise order, starting from the node on the face 1 (nodesId 1-2) and the last one referring to the central node of the element; if triangular, the 8th and 9th columns, as well as the 4th one, contain 0). The last column contains the material Id number, since the material properties are included in one more table called *materials* which contains, for each row (material kind), the μ parameter (in case it's needed more than one).

-Concerning the BCs (Boundary Conditions) a table called ***BCkind*** contains, for each row that refers to the BC Id number (there can be more than one), in the first column the letters *N* or *D*, in case of Neumann or Dirichlet BC, while the second column contains the BC value. A table called *faceBC* gives informations about what BC is imposed on the faces of the mesh: so, for each face there can be two different BCs at the same time. The first column is referred to the element Id number and the second to the face Id number of that specific element (1 for nodes 1-2 link, 2 for nodes 2-3, 3 for nodes 3-1 (or 3-4 in case of quadrilateral element), 4 for nodes 4-1 in case of quadrilateral element). The third column contains the first BC Id number (can be either N or D), and the forth one contains the possible extra BC (in the sense that if the first one is a N condition, then the second one for the same face must be a D one). If there is no extra BC the fourth cell contains 0. A last table ***nodeBC*** contains the nodal BCs, which can be only of Dirichlet kind: the first column contains the node Id number while the second one contains the BC Id number.

-The last input data refer to the shape functions that the program will use to interpolate nodal values.A table called ***shapeFunc*** contains 4 rows (in order: triangular-linear,

triangular-quadratic, quadrilateral-linear, quadrilateral-quadratic) and 8 colums (in case of simpler problems than quadrilateral-quadratic some of them can be null). In each cell there is the shape function referred to the reference element for every reference node. In ***shapeGrad***, as vectors of two components, there are the gradiente of the shape functions.
All these tables are stored inside a **structure** called *thisMesh*, following this order:
thisMesh.nodalCoords (*nodalCoordinates*)
thisMesh.elementInfos (*elementInfos*)
thisMesh.faceBC (*faceBC*)
thisMesh.nodeBC (*nodeBC*)
thisMesh.shapeFunc (*shapeFunc*)
thisMesh.shapeGrad (*shapeGrad*)

2) **ELEMENT MATRIX**

*LOOP "FOR" ON EACH ELEMENT (NoElem = no. of rOws in **thisMesh.elementInfos**)*

---

gaussianQuadrature

INPUT: ***thisMesh.elementInfos***
OUTPUT: ***GaussPointWeight - GaussPointLocation - GaussNoPoints***

Description: checks which problem is needed to be solved (ex.: "triangle-linear"), and gives as outputs informations about the ***gaussianQuadrature***

Uses: none
Used by: ***jacobianMatrix - derivatedShapeFunctions - elementMatrix***

Comments:
Possible element shape are:
1. Triangles
2. Quadrilaterals
Possible kinds of interpolation problems are:
1. Linear
2. Quadratic

---

jacobianMatrix

INPUT: ***GaussPointLocation - thisMesh.nodalCoords - thisMesh.elementInfos***

OUTPUT: ***JacobianMatrix - DetJacobianMatrix***

Description: gets informations about the gaussian points locations inside the reference element and evaluates the Jacobian and its determinant, by knowing the internal coordinates of the nodes of the element. Then evaluates this two quantities in gaussian points.

Uses: ***gaussianQuadrature***
Used by: ***elementMatrix***

Comments:
Already knows what kind of problem is solving because has informations from ***gaussianQuadrature***

---

derivatedShapeFunctions

INPUT: ***thisMesh.shapeGrad - GaussPointLocation***
OUTPUT: ***DerShapeFunctions***

Description: evaluates the position of the gauss points, and, knowing the kind of problem, gets from the table ***thisMesh.shapeGrad*** all the gradients of the shape functions linked to the reference nodes. Finally calculates the values on the Gauss points.

Uses: ***gaussianQuadrature***
Used by: ***elementMatrix***

Comments:
none

---

elementMatrix

INPUT: ***thisMesh.elementInfos - GaussPointWeight - Jacobian Matrix - DetJacobianMatrix - DerShapeFunction - GaussNoPoints***
OUTPUT: ***elementStiffnessMatrix***

Description: making a loop on integration points, calculates the contribution to the element stiffness matrix and sum all of them. From ***thisMesh.elementInfos***, gets informations about the material through the μ parameter.

Uses: ***gaussianQuadrature - jacobianMatrix - derivatedShapeFunctions***

Used by: ***assemblyProcess***

Comments:
none

---

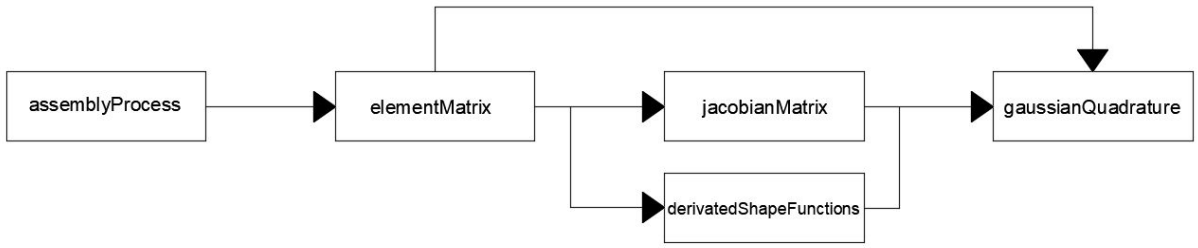
assemblyProcess

INPUT: ***elementStiffnessMatrix - thisMesh.elementInfos - thisMesh.nodalCoords***
OUTPUT: ***globalStiffnessMatrix***

Description: gets informations about the location of the element and creates the assembly by using the matricial product T^TKT and adds the contribution to the global matrix element by element.
Uses: ***elementMatrix***
Used by: none

Comments:
none

END OF LOOP ON ELEMENTS



3) **BOUNDARY CONDITIONS**

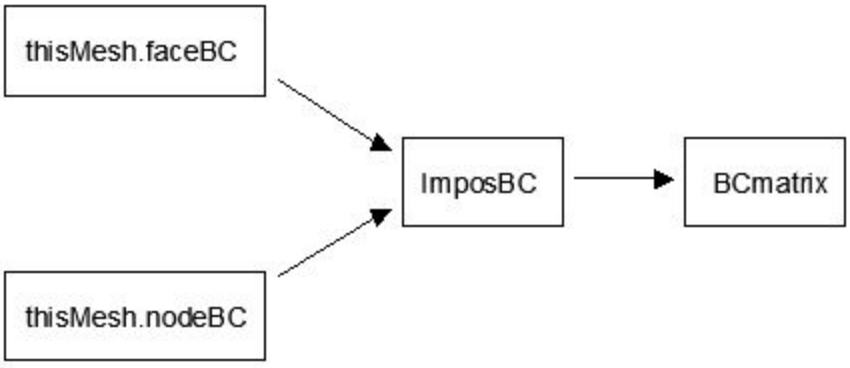
ImposBC(chose either Dirichlet or Neumann Boundary Conditions)

In order to solve the system, we need to impose the suitable boundary conditions including Dirichlet and Neumann Boundary Conditions.

INPUT:***thisMesh.faceBC-thisMesh.nodeBC***
OUTPUT: the ***BCmatrix***

Description: The function ***imposBC*** can impose the boundary conditions on the corresponding element nodes and element faces and then give the ***BCmatrix*** which can be be used in solving the equation system.

Uses:***thisMesh.faceBC-thisMesh.nodeBC***
Used by: ***SolvingSystem***



---

4) **SOLVING THE SYSTEM OF EQUATIONS**

After imposing the boundary conditions, we can get the system of equations which could be solved by numerical solutions like direct method or iterative method.

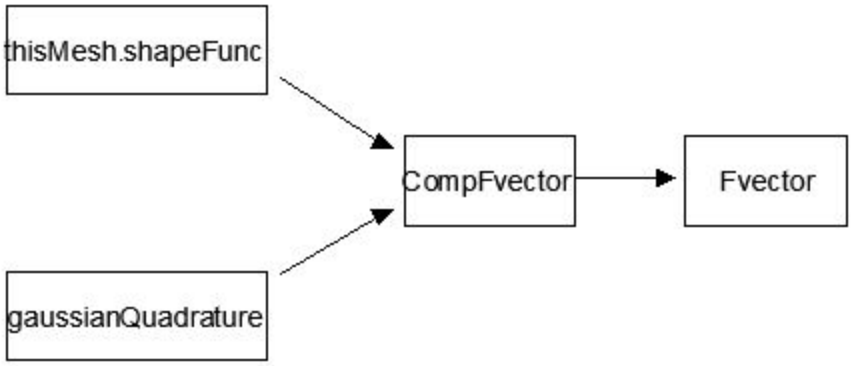
CompFvector

We have already got the ***globalStiffnessMatrix,*** the next step is to compute the RHS of the equation. And it is ***CompFvector.***

INPUT: ***thisMesh.shapeFunc-gaussianQuadrature***
OUTPUT:***Fvector***

Uses:***thisMesh.shapeFunc-GaussPointWeight - GaussPointLocation - GaussNoPoints***
Used by: ***SolvingSystem***

Description: The function ***ComFvector*** can compute the RHS of the system of equations.

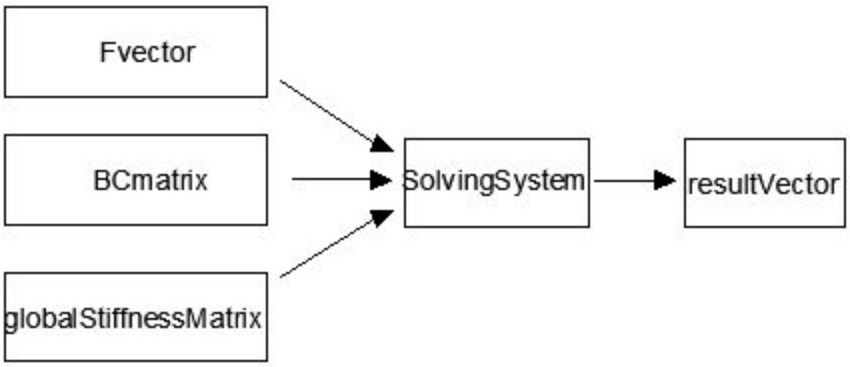


SolvingSystem

After getting the ***globalStiffnessMatrix***, ***Fvector*** and imposing the boundary conditions, we have completed building the system of equations.

INPUT:***globalStiffnessMatrix-Fvector-BCmatrix***
OUTPUT:***resultVector***

Description: After solving the system of equations, we get the final result ***resultVector.***



---

5) **FINAL PROPOSE**

Our program can solve the problems of 1D and 2D. And we can extend our program to make it capable of solving the 3D problems. We need to extend the coordinates, the gaussianQuadrature, the shape functions and the method used to solve the system of equations to make it more efficient.