Programming for Engineers and Scientists
-
FEM Matlab code design to solve Poisson's equation

By
Ahmed Saeed Sherif
Chiluba Isaiah Nsofu
Nikhil Dave

February 24, 2018

1 Introduction

The work presented in this report is concerned with designing a Finite Element Method (FEM) code in Matlab to solve 2D Poisson's equation. One of the main considerations is to generalize the code for using elements of different shapes and orders. Apart from that, the code structure relies heavily on the use of functions which makes it readable and easily editable. Firstly, an overview of the proposed design is presented as a flowchart where the main sections of the codes are shown, followed by a brief explanation. Secondly, a dependence graph of the functions to be used is presented with the objective to understand the reliance of each function in the potential code. This is followed by a library of functions where each function is described and their inputs and outputs are listed and explained. Next, a section explaining data structures is included and finally a reference to the upcoming tasks is presented.

2 FEM code flowchart

2.1 Flowchart shapes and colour code description

To begin our discussion on the flowchart shown in figure 1, we start with the description of each of the blocks in the chart. As it can be seen in the flowchart we have decided to use different shapes for clarity to represent different parts of the code. Therefore, each shape represents a specific function that performs a certain task described by its contents.

Diamond : This shape is used to represent the decision point within the code. For example the decision point at the end of the Gauss iteration loop, element iteration loop and the comparison of the FEM solution with the analytical solution .

Hexagon : Highlighted in yellow, this shape is used to indicate the starting point of a loop. Specific examples been the loop across the gauss points in an element and also a loop for all the elements in the mesh.

Parallelogram : This shape is used to represent the load of data such as the reference elements and eventually mesh related data.

Circle: This shape is used to represent the start and the end of program. For example the entire process of defining all the basic definitions at the beginning of the code.

Rectangle with rounded-corners : These shapes represents a main process such as the assembling of the local and global stiffness matrices.

Rectangle with right angled-corners : This is used to represent the parts of the code such as the process, plot variables and compute errors.

2.2 Specific code structure divisions

Just like any formidable finite element code, we have decided to partition the entire code structure into three different sections, these being the pre-process, process and post-process sections.
The **pre-process** section is made up of blocks denoted by the following;

- Basics definitions
- Load reference element
- Load mesh data

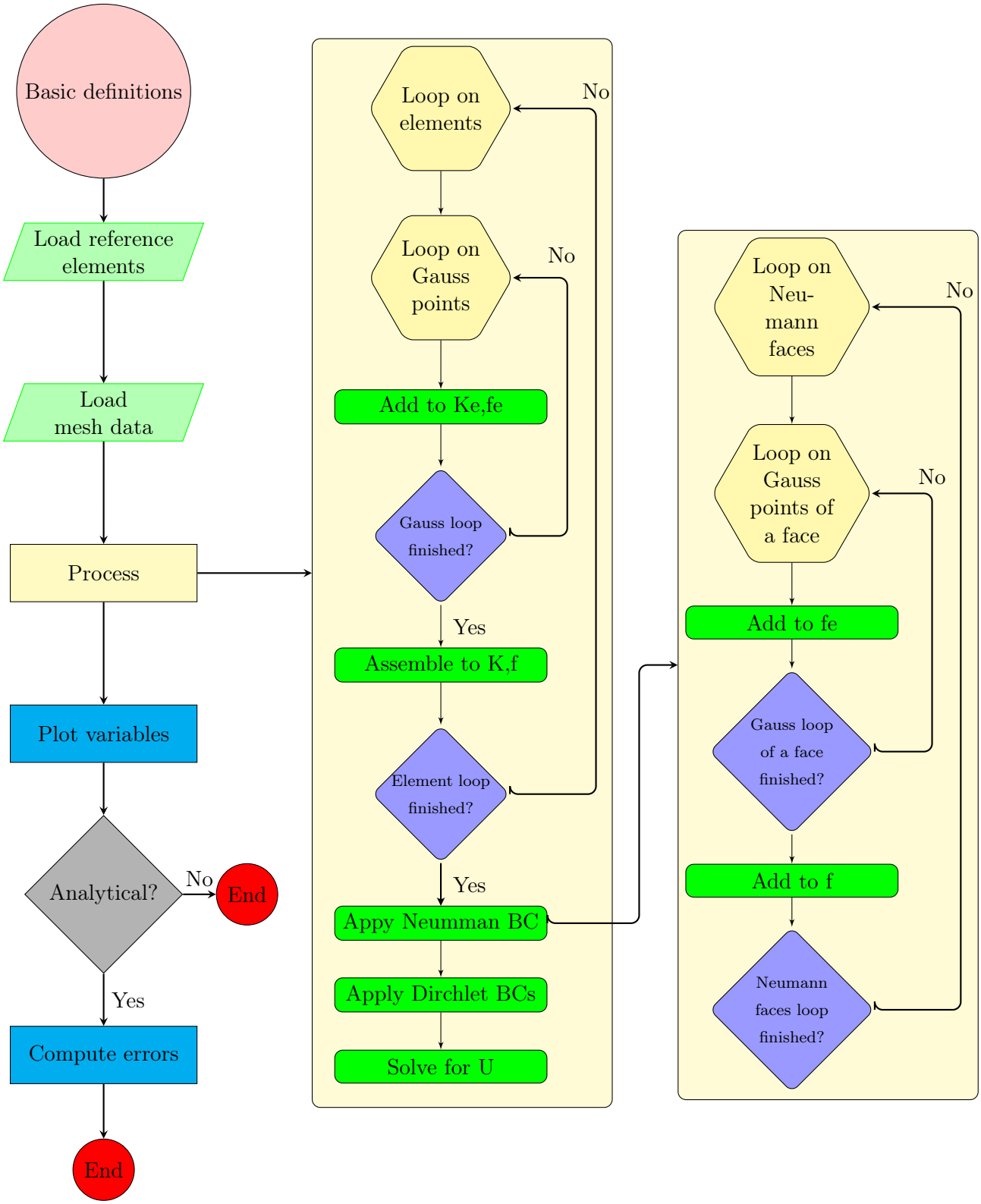

Figure 1: Flowchart of the FEM code

The **process** section is made up of the blocks represented by the following;

- Loop on elements
- Loop on Gauss points
- Add to Ke, fe
- Gauss loop finished
- Assembly to K,F
- Element loop finished
- Apply Neumann BC
- Apply Dirchlet BC
- Solve for U

The **post-process** section is made up of the blocks represented by the following;

- Plot variables
- Compute errors

2.2.1 Pre-process section

This is indeed the starting point of the code. Within this section we will define functions that will be able to load in the basic definitions of the problem in accordance with the user specifications. For example when it comes to solving the 2D Poisson's equation the user will be asked to select the type of element and the interpolation degree to be used. Upon doing that, the element and nodal data pertaining to the selected reference element will be loaded into the code. This data includes loading of the elemental coordinates, shape functions, shape functions derivatives and also the number of Gauss points and weights that will be used for the numerical integration. It is also within this section that the mesh will be loaded hence the user will have an option to choose the level of mesh refinement. Once the mesh has been loaded, the information concerning the connectivity and coordinates matrices that will be extracted using a function.

2.2.2 Process section

This section describes another important part of the FEM code. It is within this section where tasks such as Gauss integration and calculation of the unknown solution is performed. Once the much needed information has been received from the functions in the pre-process section, the first thing that will be carried out is the formation of the local stiffness (Ke) matrix and R.H.S (fe) vector in each element. To perform this task we will need to perform the Gauss integration using the Gauss points in each specific element. After the Gauss integration process is finished we will begin the assembly of the global stiffness matrix and R.H.S vector. This process will be carried out for all the elements in the mesh. Once this process has finished we will then apply the boundary conditions on the specified parts on the domain which will then allow us to solve the entire system for the unknown solution.

2.2.3 Post-process section

It is within this section that once the model has been solved, the results are visualized and analyzed. Another important function is to compute errors for problems with known analytical solution. For this, a mesh convergence study is performed to check if the optimum convergence rates are achieved.

3 Functions dependence graph

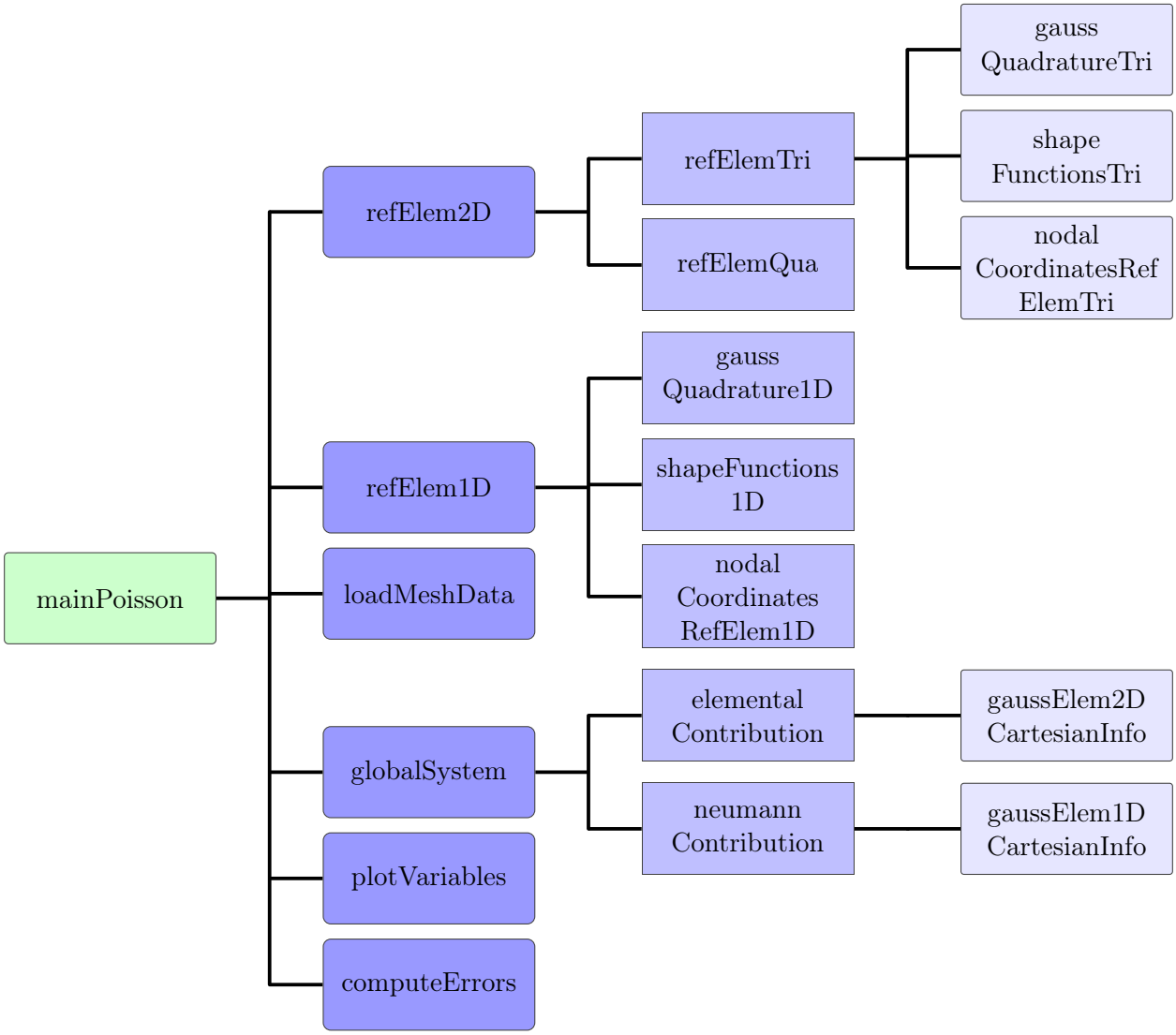

Figure 2: Dependence graph of the FEM code functions

Figure 2 shows the dependence graph of the functions to be used while writing the Finite Element Method code in MATLAB. The objective of designing a dependence graph is to understand the reliance of each function of the potential code. For the purpose of clarity and legibility of the process, most processes involved in the design are defined as independent functions which are to be called from the *mainPoisson.m* file to perform the action and output the results to the main file. This design gives us the flexibility to incorporate any required change to the code much easily.

To launch the problem solver, we run the script *mainPoisson.m* which comprises of the core structure to be executed. After the user provides the input data regarding the type and order of the mesh element, the functions related to the reference element (*refElem2D* and *refElem1D*) are called. For instance, in case of triangular elements, the procedure is as follows:

- Firstly, determine the number of Gauss points required for numerical integration.
- Load the corresponding Gauss points and weights using function *gaussQuadratureTri*.
- Load the shape functions and their derivatives evaluated at Gauss points as well as nodal coordinates of the reference element, using functions *shapeFunctionTri* and *nodalCoordinatesRef*.

Next, the user loads the mesh data i.e. connectivity matrices, nodal coordinates and boundary conditions data based on the level of mesh refinement using the function *loadMeshData*.

The solution process is then executed,

- Using the *globalSystem* function where a loop over elements as well as a loop over Neumann faces are performed and their contributions are assembled into the global system. In each loop iteration over elements, the elemental data are extracted and the function *elementalContribution* is called. While in each loop iteration over Neumann faces, the face data are extracted and the function *neumannContribution* is called. Finally, a reduced system of equations is obtained after applying the Dirchlet boundary conditions.

- The function *elementalContribution* performs a loop over the Gauss points of the element to compute K_e, f_e. In each loop iteration, all the data evaluated at the Gauss point are extracted using the function *gaussElem2DCartesianInfo*.

- *gaussElem2DCartesianInfo* extracts the values of the shape functions, their derivatives, Gauss weight and physical coordinates of the Gauss point as well as the Jacobian and its determinant evaluated at a specific Gauss point.

- The function *neumannContribution* performs a loop over the Gauss points of the Neumann face to compute and add the Neumann boundary contribution to f_e. In each loop iteration, all the data evaluated at the Gauss point are extracted using the function *gaussElem1DCartesianInfo*.

- *gaussElem1DCartesianInfo* extracts the values of the shape functions, the outward normal unit vector, Gauss weight and physical coordinates of the Gauss point as well as the determinant of the Jacobian evaluated at a specific Gauss point.

After the reduced system of equations is solved, the solution variables are plotted and error computation is performed using *plotVariables* and *computeErrors* functions, respectively.

4 Library of functions

This section helps to further understand the routines performed by each block in the dependence graph. The functions are explained in more details, where the inputs and outputs for each function are listed and commented.

refElem2D
Inputs: `elementType`, `elementDegree`
Outputs: `refElem`
Description: Gives all the data associated to a 2D reference element.
Uses: `refElemTri`, `refElemQua` **Used by:** `mainPoisson`

Comments:
 	
Valid inputs:
`elementType`: 0 for quadrilaterals or 1 for triangles.
`elementDegree`: 1 for linear elements or 2 for quadratic elements.
 	
Outputs:
`refElem` is a data structure containing:
`refElem.gaussPoints`
`refElem.gaussWeights`
`refElem.nOfGauss`
`refElem.nOfNodes`
`refElem.shapeFunctions`
`refElem.coordinates`
`refElem.elementType`
`refElem.elementDegree`

refElemTri
Inputs: `elementDegree`
Outputs: `refElem`
Description: Gives all the data associated to a triangular reference element.
Uses: `gaussQuadratureTri`, `shapeFunctionsTri`, `nodalCoordinatesRefElemTri` **Used by:** `refElem2D`

Comments:
 	
Valid inputs:
`elementDegree`: 1 for linear triangles or 2 for quadratic triangles.
 	
Outputs:
`refElem` is a data structure containing:
`refElem.gaussPoints`
`refElem.gaussWeights`
`refElem.nOfGauss`
`refElem.nOfNodes`
`refElem.shapeFunctions`
`refElem.coordinates`

refElemQua
Inputs: `elementDegree`
Outputs: `refElem`
Description: Gives all the data associated to a quadrilateral reference element.
Uses: `gaussQuadratureQua`, `shapeFunctionsQua`, `nodalCoordinatesRefElemQua` **Used by:** `refElem2D`

Comments:
 	
Valid inputs:
`elementDegree`: 1 for linear quadrilaterals or 2 for quadratic quadrilaterals.
 	
Outputs:
`refElem` is a data structure containing:
`refElem.gaussPoints`
`refElem.gaussWeights`
`refElem.nOfGauss`
`refElem.nOfNodes`
`refElem.shapeFunctions`
`refElem.coordinates`

gaussQuadratureTri
Inputs: `elementDegree`
Outputs: `gaussPoints`, `gaussWeights`
Description: Implements the Gauss quadrature in a reference triangle to be used in integrals' evaluation.
Uses: none **Used by:** `refElemTri`
Comments: Valid inputs: `elementDegree`: 1 for linear triangles or 2 for quadratic triangles. Outputs: `gaussPoints`: Matrix of Gauss points' coordinates in the reference triangle. `gaussWeights`: Vector of Gauss weights corresponding to each Gauss point.

shapeFunctionsTri
Inputs: `elementDegree`, `gaussPoints`
Outputs: `shapeFunctions`
Description: Gives the values of the shape functions and their derivatives evaluated at the Gauss points in the reference triangle.
Uses: none **Used by:** `refElemTri`
Comments: Valid inputs: `elementDegree`: 1 for linear triangles or 2 for quadratic triangles. `gaussPoints`: Matrix of Gauss points' coordinates in the reference triangle. Outputs: `shapeFunctions` is a tensor of size `(nOfGauss, nOfNodes, 3)` where: `shapeFunctions(:,:,1)` contains the shape functions, `shapeFunctions(:,:,2)` contains the first derivative in direction of ξ, `shapeFunctions(:,:,3)` contains the first derivative in direction of η.

nodalCoordinatesRefElemTri
Inputs: `elementDegree`
Outputs: `coordinates`
Description: Gives the coordinates of the nodes in the reference triangle.
Uses: none **Used by:** `refElemTri`
Comments: Valid inputs: `elementDegree`: 1 for linear triangles or 2 for quadratic triangles. Outputs: `coordinates`: Matrix of nodal coordinates of the reference triangle.

refElem1D
Inputs: `elementDegree`
Outputs: `refFace`
Description: Gives all the data associated to a 1D reference element. The output data are used for computing the integrals on Neumann boundaries.
Uses: `gaussQuadrature1D`, `shapeFunctions1D`, `nodalCoordinatesRefElem1D` **Used by:** `mainPoisson`

Comments:
 	
Valid inputs:
`elementDegree`: 1 for linear elements or 2 for quadratic elements.
 	
Outputs:
`refFace` is a data structure containing:
`refFace.gaussPoints`
`refFace.gaussWeights`
`refFace.nOfGauss`
`refFace.nOfNodes`
`refFace.shapeFunctions`
`refFace.coordinates`
`refFace.elementDegree`

gaussQuadrature1D
Inputs: `elementDegree`
Outputs: `gaussPoints`, `gaussWeights`
Description: Implements the Gauss quadrature in the reference 1D element.
Uses: none **Used by:** `refElem1D`
Comments: Valid inputs: `elementDegree`: 1 for linear triangles or 2 for quadratic triangles. Outputs: `gaussPoints`: Vector of Gauss points' coordinates in the reference 1D element. `gaussWeights`: Vector of Gauss weights corresponding to each Gauss point.

shapeFunctions1D
Inputs: `elementDegree`, `gaussPoints1D`
Outputs: `shapeFunctions`
Description: Gives the values of the shape functions and their derivatives evaluated at the Gauss points in the reference 1D element.
Uses: none **Used by:** `refElem1D`
Comments: Valid inputs: `elementDegree`: 1 for linear triangles or 2 for quadratic triangles. `gaussPoints`: Vector of Gauss points' coordinates in the reference 1D element. Outputs: `shapeFunctions` is a tensor of size `(nOfGauss, nOfNodes, 2)` where: `shapeFunctions(:,:,1)` contains the shape functions, `shapeFunctions(:,:,2)` contains the first derivative.

nodalCoordinatesRefElem1D
Inputs: `elementDegree`
Outputs: `coordinates`
Description: Gives the coordinates of the nodes in the reference 1D element.
Uses: none **Used by:** `refElem1D`
Comments: Valid inputs: `elementDegree`: 1 for linear triangles or 2 for quadratic triangles. Outputs: `coordinates`: Vector of nodal coordinates of the reference 1D element.

loadMeshData
Inputs: `meshFile`
Outputs: `mesh`
Description: Loads all the data related to a mesh such as the nodal Cartesian coordinates, connectivity table, Dirchlet and Neumann boundary conditions data. This function loads the data from a (.DAT) file.
Uses: none **Used by:** `mainPoisson`
Comments: Valid inputs: `meshFile`: takes the format `domainShape_elementType_elementDegree_meshRefinementLevel` for example: `rectangle_Tri_P1_H1` Outputs: `mesh` is a data structure containing: `mesh.X`: Matrix of nodal coordinates of the mesh. `mesh.T`: Table of connectivities. `mesh.DBC`: Matrix of size `(nNodesDBC,2)` containing Dirchlet nodes with their prescribed values. `mesh.NBC`: Matrix of size `(nFacesNBC,nOfFaceNodes+1)` containing the global number of face nodes and an index corresponding to a Neumann boundary function.

globalSystem

Inputs: `refElem`, `refFace`, `mesh`

Outputs: `K`, `f`

Description: This function performs the loop over elements and assemble the elemental contributions to the global stiffness matrix and forcing vector, then gives the reduced system of equations after applying Dirchlet boundary conditions.

Uses: `elementalContribution`, `neumannContribution`
Used by: `mainPoisson`

Comments:

Valid inputs:

Data structure `refElem` contains:
`refElem.gaussPoints`
`refElem.gaussWeights`
`refElem.nOfGauss`
`refElem.nOfNodes`
`refElem.shapeFunctions`
`refElem.coordinates`
`refElem.elementType`
`refElem.elementDegree`

Data structure `refFace` contains:
`refFace.gaussPoints`
`refFace.gaussWeights`
`refFace.nOfGauss`
`refFace.nOfNodes`
`refFace.shapeFunctions`
`refFace.coordinates`
`refFace.elementDegree`

Data structure `mesh` contains:
`mesh.X`: Matrix of nodal coordinates of the mesh.
`mesh.T`: Table of connectivities.
`mesh.DBC`: Matrix of Dirchlet BCs data.
`mesh.NBC`: Matrix of Neumann BCs data.

Outputs:
`K`: Global stiffness matrix after applying DBC.
`f`: Global forcing vector with DBC and NBC contributions.

elementalContribution
Inputs: `refElem`, `Xe`, `Te`
Outputs: `Ke`, `fe`
Description: This function performs a loop over Gauss points of an element and assemble the Gauss points contributions to the elemental stiffness matrix.
Uses: `gaussElem2DCartesianInfo` **Used by:** `globalSystem`

Comments:
 	
Valid inputs:
Data structure `refElem` contains:
`refElem.gaussPoints`
`refElem.gaussWeights`
`refElem.nOfGauss`
`refElem.nOfNodes`
`refElem.shapeFunctions`
`refElem.coordinates`
`refElem.elementType`
`refElem.elementDegree`
 	
`Xe`: Matrix of nodal coordinates of an element.
`Te`: Table of connectivities of an element.
 	
Outputs:
`Ke`: Elemental stiffness matrix.
`fe`: Elemental forcing vector with source term contributions.

neumannContribution
Inputs: `refFace`, `mesh.NBC`
Outputs: `fe`
Description: This function performs a loop over Gauss points of a Neumann face and assemble the Gauss points contributions to the elemental forcing vector.
Uses: `gaussElem1DCartesianInfo` **Used by:** `globalSystem`

Comments:
 	
Valid inputs:
Data structure `refFace` contains:
`refFace.gaussPoints`
`refFace.gaussWeights`
`refFace.nOfGauss`
`refFace.nOfNodes`
`refFace.shapeFunctions`
`refFace.coordinates`
`refFace.elementDegree`
 	
`mesh.NBC`: Matrix of Neumann BCs data.
 	
Outputs:
`fe`: Elemental forcing vector with NBC contributions.

gaussElem2DCartesianInfo

Inputs: `refElem`, `Xe`, `iGauss`

Outputs: `N`, `dNx`, `detJ`, `w`, `Xg`

Description: Gives all the functions evaluated at a specific Gauss point in a 2D element.

Uses: none
Used by: `elementalMatrices`

Comments:

Valid inputs:
Data structure `refElem` contains:
`refElem.gaussWeights`
`refElem.shapeFunctions`

`Xe`: Matrix of nodal coordinates of the physical element.
`iGauss`: index of Gauss point.

Outputs:
`N`: Shape functions evaluated at `iGauss`.
`dNx`: First derivatives with respect to Cartesian coordinates evaluated at `iGauss`.
`detJ`: Determinant of the Jacobian evaluated at `iGauss`.
`w`: Gauss weight associated to `iGauss`.
`Xg`: Cartesian coordinates of `iGauss` to be used for source term evaluation.

gaussElem1DCartesianInfo

Inputs: `refFace`, `Xf`, `iGauss`

Outputs: `N`, `n`, `detJ`, `w`, `Xg`

Description: Gives all the functions evaluated at a specific Gauss point in a 1D element.

Uses: none
Used by: `elementalVectors`

Comments:

Valid inputs:
Data structure `refFace` contains:
`refFace.gaussWeights`
`refFace.shapeFunctions`

`Xf`: Matrix of nodal coordinates of the 1D physical element.
`iGauss`: index of Gauss point.

Outputs:
`N`: Shape functions evaluated at `iGauss`.
`n`: Outward unit normal vector evaluated at `iGauss`.
`detJ`: Determinant of the Jacobian evaluated at `iGauss`.
`w`: Gauss weight associated to `iGauss`.
`Xg`: Cartesian coordinates of `iGauss` to be used for Neumann BC function evaluation.

plotVariables
Inputs: `what2Plot`, `U`, `mesh`, `refElem`, `refFace`
Outputs: `fig`
Description: Plots the variables of interests
Uses: none **Used by:** `mainPoisson`
Comments: Valid inputs: `what2Plot`: `'primalVariable'`, `'secondaryVariable'` `U`: the numerical nodal solution. Outputs: `fig`: The plot of primal or secondary variable over the mesh.

computeError
Inputs: `U`, `Uexact`, `mesh`, `refElem`
Outputs: `error`
Description: Computes the error L2 and H1 norms if the analytical solution of the problem exists.
Uses: none **Used by:** `mainPoisson`

Comments:
 	
Valid inputs:
`U`: the numerical nodal solution.
`Uexact`: the analytical nodal solution.
 	
Outputs:
Data structure `error` contains:
`error.L2`
`error.H1`
`error.L2elem`

5 Data structures

It is very useful to put together all the related data using data structures. A simple example, for instance, is gathering all the data related to the mesh in one data structure. In this design of the Matlab code, three data structures are used: `mesh`, `refElem`, and `refFace`.

5.1 First data structure: `mesh`

The data structure `mesh` is defined as follows:

- `mesh.X`: Matrix of nodal coordinates of size (number of nodes, spatial dimension). Therefore, for a 2D mesh with n number of nodes, it is stored as:

$$\texttt{mesh.X} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

- `mesh.T`: Matrix representing the table of connectivities of size (number of elements, number of nodes per element + 1). Thus for a mesh with n_{el} linear triangles, the matrix `mesh.T` is of size (n_{el}, 4), and is stored as:

$$\texttt{mesh.T} = \begin{bmatrix} \text{elem 1-node 1} & \text{elem 1-node 2} & \text{elem 1-node 3} & \text{elem 1-material index} \\ \vdots & \vdots & \vdots & \vdots \\ \text{elem } n_{el}\text{-node 1} & \text{elem } n_{el}\text{-node 2} & \text{elem } n_{el}\text{-node 3} & \text{elem } n_{el}\text{-material index} \end{bmatrix}$$

- `mesh.DBC`: Matrix storing the Dirchlet boundary conditions data. It is of size (number of Dirchlet nodes, 2) where the first column contains the global number of the node while the second column contains the corresponding prescribed value of the primal variable. A mesh with n_{DBC} nodes on a Dirchlet boundary is stored as:

$$\texttt{mesh.DBC} = \begin{bmatrix} \text{DBC 1-node} & \text{DBC 1-value} \\ \vdots & \vdots \\ \text{DBC } n_{DBC}\text{-node} & \text{DBC } n_{DBC}\text{-value} \end{bmatrix}$$

- `mesh.NBC`: Matrix storing the Neumann boundary conditions data. It is of size (number of Neumann faces, number of nodes per face + 1). Considering a mesh of linear triangles with n_{NBC} faces on Neumann boundary, the matrix `mesh.NBC` will be of size (n_{NBC}, 3). Each row of the matrix corresponds to a face on a Neumann boundary, while the first and second columns correspond to the global number of the nodes on that face. The third column contains an index that corresponds to a Neumann boundary function.

$$\texttt{mesh.NBC} = \begin{bmatrix} \text{NBC 1-node 1} & \text{NBC 1-node 2} & \text{NBC 1-function index} \\ \vdots & \vdots & \vdots \\ \text{NBC } n_{NBC}\text{-node 1} & \text{NBC } n_{NBC}\text{-node 2} & \text{NBC } n_{NBC}\text{-function index} \end{bmatrix}$$

It is important to note that a function index is stored instead of a value in `mesh.NBC`, unlike `mesh.DBC`. The reason is that in case of variable flux, the function is evaluated at Gauss points when the Neumann boundary integral is computed. Moreover, it is also important to note that the output of the Neumann function is a vector of size (spatial dimension).

5.2 Second data structure: `refElem`

The data structure `refElem` is defined as follows:

- `refElem.nOfGauss`: Number of Gauss points to be used in integrals evaluation.

$$\texttt{refElem.nOfGauss} = n_{gp}$$

- `refElem.gaussPoints`: Matrix of Gauss points' coordinates in the reference element.

$$\texttt{refElem.gaussPoints} = \begin{bmatrix} \xi_{gp1} & \eta_{gp1} \\ \vdots & \vdots \\ \xi_{n_{gp}} & \eta_{n_{gp}} \end{bmatrix}$$

- `refElem.gaussWeights`: Vector of Gauss weights corresponding to each Gauss point.

$$\texttt{refElem.gaussWeights} = \begin{Bmatrix} \texttt{w}_{gp1} \\ \vdots \\ \texttt{w}_{n_{gp}} \end{Bmatrix}$$

- `refElem.nOfNodes`: Number of nodes per element.

$$\texttt{refElem.nOfNodes} = n_{en}$$

- `refElem.shapeFunctions`: is a tensor of size `(nOfGauss, nOfNodes, 3)` containing all the shape functions and their derivatives evaluated at all the Gauss points.
  `refElem.shapeFunctions(:,:,1)` contains the shape functions,
  `refElem.shapeFunctions(:,:,2)` contains the first derivative in direction of ξ,
  `refElem.shapeFunctions(:,:,3)` contains the first derivative in direction of η.

- `refElem.elementType`: The shape of the element. 0 means quadrilateral and 1 is triangle.

- `refElem.elementDegree`: The order of the element. 1 means linear and 2 is quadratic.

- `refElem.coordinates`: The nodal coordinates in the reference element.

$$\texttt{refElem.coordinates} = \begin{bmatrix} \xi_1 & \eta_1 \\ \vdots & \vdots \\ \xi_{n_{en}} & \eta_{n_{en}} \end{bmatrix}$$

5.3 Third data structure: `refFace`

In case of 2D problems, the data structure `refFace` corresponds to a 1D reference element. The definition of this structure is very similar to that of `refElem` with minor changes given as follows:

- `refFace.gaussPoints`: Vector of Gauss points' coordinates in the reference 1D element.

$$\texttt{refFace.gaussPoints} = \begin{Bmatrix} \xi_{gp1} \\ \vdots \\ \xi_{n_{gp}} \end{Bmatrix}$$

- `refFace.coordinates`: The nodal coordinates in the reference 1D element.

$$\texttt{refFace.coordinates} = \begin{Bmatrix} \xi_{1} \\ \vdots \\ \xi_{n_{en}} \end{Bmatrix}$$

- `refFace.shapeFunctions`: is a tensor of size `(nOfGauss, nOfNodes, 2)` containing all the shape functions and their derivatives evaluated at all the Gauss points.
  `refFace.shapeFunctions(:,:,1)` contains the shape functions,
  `refFace.shapeFunctions(:,:,2)` contains the first derivative in direction of ξ.

There is no field for `refFace.elementType` because there is only one type of elements in 1D, which is a line.

5.4 Fourth data structure: `error`

The data structure `error` is defined as follows:

- `error.L2`: a scalar number representing the L2-norm of error in the whole domain.
- `error.H1`: a scalar number representing the H1-norm of error in the whole domain.
- `error.L2elem`: a vector containing the L2-norm of error of each element.

6 Upcoming tasks

Our objective was to design a Matlab code to solve Poisson's problem. The effort has been made to make a generic design in order to account for different types of boundary conditions and source terms which are given by either constant or variable functions.

After completing the design of the FEM Matlab code, the next step is to start writing the main script of the code in Matlab and to implement the needed functions to solve the problem. It is inevitable the modifications that will be done to this proposed design during the process of developing the code.