Programming for Engineers and Scientists
FEM C++ code design to solve Poisson's equation

By
Ahmed Saeed Sherif (ahmedsaeed_713@yahoo.com)
Chiluba Isaiah Nsofu (chilubansofu@hotmail.com)
Nikhil Dave (nikdave14@gmail.com)

May 13, 2018

1 Introduction

The work presented in this report is concerned with the designing of a 2D/3D Finite Element Method (FEM) code in C++. One of the main considerations in coming up with the current design is to utilise the unique aspects of C++ programming language while taking into consideration the various relationships used in the finite element method. With respect to the aforementioned goals the entire report is structured as follows. Firstly, an overview of a FEM code is presented as a flowchart in Figure 1 where the main sections of the codes are highlighted and briefly explained. Secondly, the designed classes are proposed and the member functions to be used are presented with the objective to understand the reliance of each class in the potential code. This is followed by a library of classes where each class is described and their inputs and outputs are listed and explained. Next, a section explaining the good/bad properties of the design, alternative possibilities, is included and finally a reference to the upcoming tasks is presented.

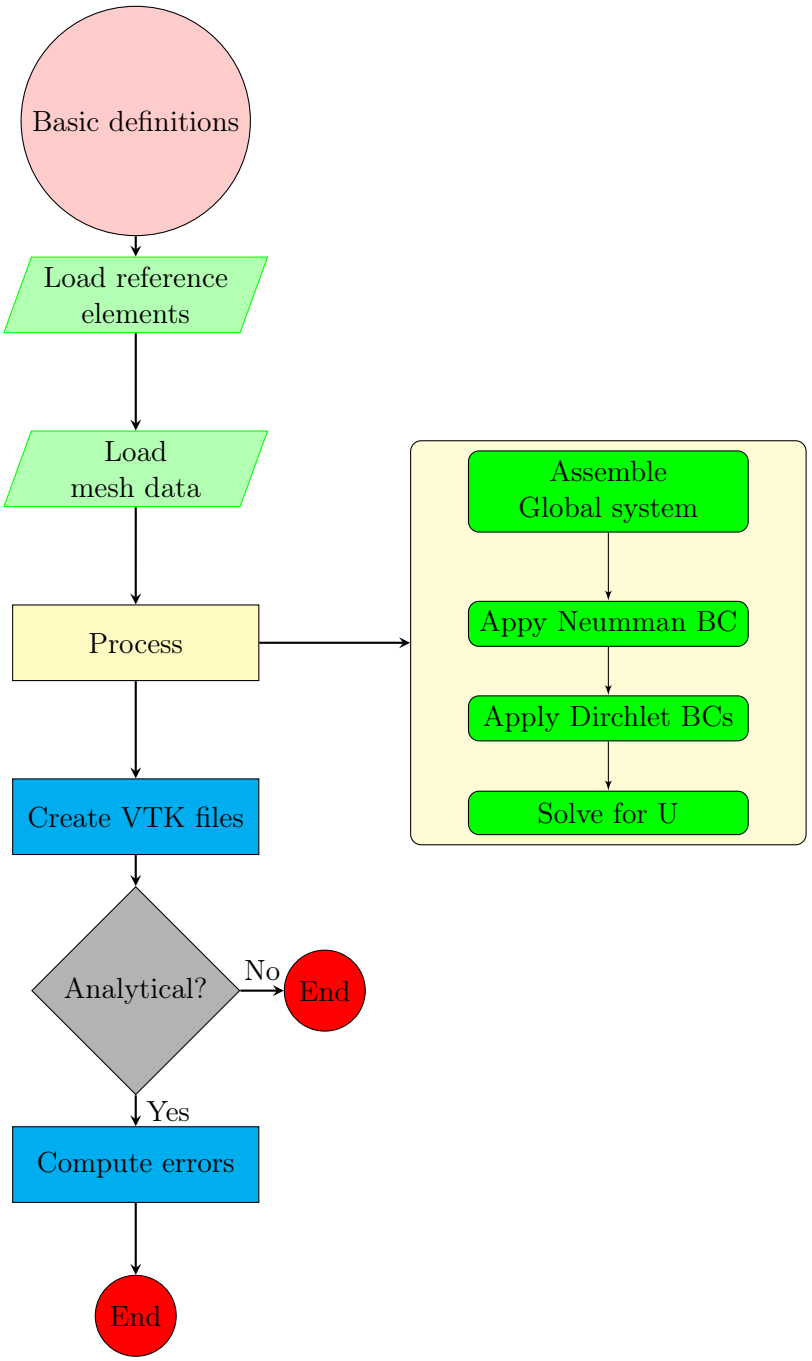

Figure 1: Flowchart of the FEM code

2 Design of classes

Figures 2 shows the graph of the potential classes that will be used in the implementation of the Finite Element Method code. In the proposed design, the class `Problem` has member functions that provides the material properties, source term, Neumann flux and exact solution (if it exists). Secondly, the class `Mesh` defines an object that has the mesh data such as nodal coordinates, table of connectivities, Neumann and Dirichlet boundary data. The reference element is constructed using two different classes, `GaussQuadrature` and `ShapeFunctions`, where the former provides an object that has integration points and weights and the latter provides the shape functions and their derivatives evaluated at Gauss points. The class `SytemSolve` defines an object that has the nodal solution, the global system and the right hand side forcing vector. Hence, this class has member functions that assemble, apply boundary conditions and solve the linear system. Finally, all the aspects related to the post-processing section are defined by an object of type `PostProcess`.

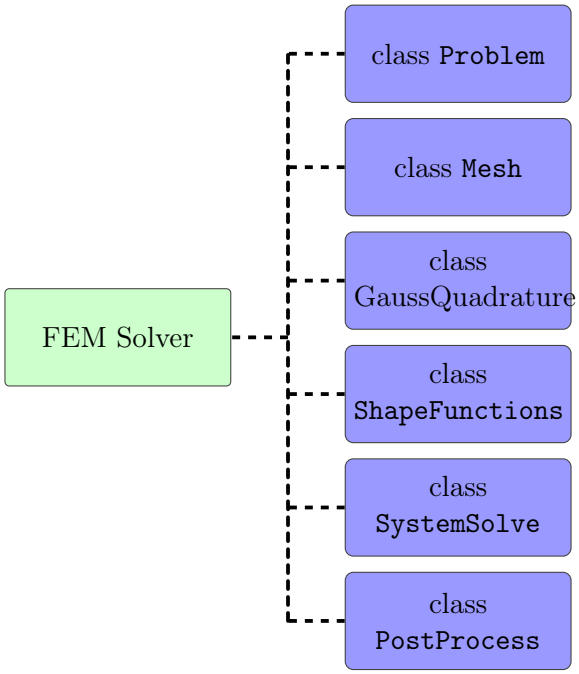

Figure 2: Designed classes for the Poisson's FEM solver

All the details of the designed classes including member data and methods are presented in the next section.

3 Implemented Classes

class **Mesh**

Data Members:
`X`
`T`
`DBC`
`NBC`

Methods: `Mesh::loadMeshData(nsd, elemType, elemDegree, H, meshFile)`
Comments: `X`: is a matrix containing mesh nodal coordinates `T`: is a matrix containing the connectivity table and the material indices `DBC`: is a matrix containing Dirchlet boundary information `NBC`: is a matrix containing Neumann boundary information `nsd`: spatial dimension `elemType`: 0 for Line(1D)/Quad(2D) or 1 for Tri(2D)/Tet(3D) `elemDegree`: 1 for linear and 2 for quadratic `H`: Level of mesh refinement `meshFile`: the name of the file containing the mesh data

class **GaussQuadrature**

Data Members:
`gaussPoints`
`gaussWeights`
`nOfGauss`
`nOfNodes`
`coordinates`
`elementType`
`elementDegree`
`nsd`

Methods:
`GaussQuadrature::GaussQuadrature(nsd, elementType, elementDegree)`
`GaussQuadrature::~GaussQuadrature()`

Comments: `gaussPoints`: Matrix containing coordinates of Gauss points in the reference element `gaussWeights`: Vector of Gauss corresponding Gauss weights `nOfGauss`: number of Gauss points `nOfNodes`: Number of nodes within an element `coordinates`: Matrix containing the nodal coordinates of the reference element `elementType`: 0 for Line(1D)/Quad(2D) or 1 for Tri(2D)/Tet(3D) `elementDegree`: 1 for linear and 2 for quadratic `nsd`: spatial dimension

class **ShapeFunctions**

Data Members:
`N`
`Nxi`
`Neta`
`Nzeta`

Methods:
`ShapeFunctions::ShapeFunctions(GaussQuadrature, elementType, elementDegree)`
`ShapeFunctions::~ShapeFunctions()`

Comments: `N`: Matrix containing shape functions evaluated at Gauss points `Nxi`: Matrix containing derivatives of shape functions w.r.t xi evaluated at Gauss points `Neta`: Matrix containing derivatives of shape functions w.r.t eta evaluated at Gauss points `Nzeta`: Matrix containing derivatives of shape functions w.r.t zeta evaluated at Gauss points

class **SystemSolve**

Data Members:
`U`
`K`
`F_source`
`F_neumann`
`K_reduced`
`F_reduced`

Methods:
`SystemSolve::globalSystem(GaussQuadrature, Mesh)`
`SystemSolve::applyNeumann(GaussQuadrature, Mesh)`
`SystemSolve::applyDirchlet(SystemSolve, Mesh)`
`SystemSolve::linearSystemSolve(SystemSolve)`

Comments: `U`: is the FEM solution (vector of nodal values) `K`: is the assembled stiffness matrix `F_source`: is the RHS forcing vector due to source term contribution `F_neumann`: is the RHS forcing vector due to Neumann boundary contribution `K_reduced`: is the reduced global stiffness matrix after eliminating the rows and columns corresponding to Dirchlet nodes `F_reduced`: is the reduced RHS forcing vector after eliminating the rows corresponding to Dirchlet nodes and adding the Dirchlet boundary contribution

class **postProcess**

Data Members:
`U`
`Ux`
`U_exact`
`Ux_exact`
`errL2`
`errH1`

Methods:
`postProcess::computeGradients(GaussQuadrature, SystemSolve)`
`postProcess::createVTKfile(SystemSolve, Mesh)`
`postProcess::computeErrors(GaussQuadrature, SystemSolve, Mesh)`

Comments: `U`: is the FEM solution (vector of nodal values) `Ux`: is the numerical gradients computed at Gauss points `U_exact`: is the exact nodal solutions (if exists) `Ux_exact`: is the exact gradients at Gauss points (if exists) `errL2`: is the $\mathcal{L}_2$-norm of the error `errH1`: is the $\mathcal{H}^1$-norm of the error

class **problem**

Data Members:
`U_exact`
`k`
`s`
`fluxVec`
`normalFlux`

Methods:
`problem::materialProperties`
`problem::sourceTerm`
`problem::neumannFlux`
`problem::neumannNormalFlux`
`problem::exactSol`

Comments: `U_exact`: is the exact nodal solutions (if exists) `k`: material property depending on space `s`: source term depending on space `fluxVec`: is the flux vector `normalFlux`: is the normal flux at boundaries

4 Conclusions

In conclusion, the use of classes presents a better aspect of gathering all the related information into objects which is fundamentally a good practice. One of the features of the proposed design is that the class `mesh` would only work if the mesh files are given together with the boundary condition data, otherwise some modifications are required in order to include the boundary data in the mesh files. This might become an obstacle in solving a specific problem. A fundamental issue to consider is the implementation of a linear system solver. Currently, we are not aware of an available linear solver which might be useful while implementing the proposed design in C++. The next step in this module is the actual implementation of the code in C++. Depending on the problems we might face during the implementation, we expect to make a few changes in the current design and report it in the next report of this assignment.