250995 - Programming for Engineers and Scientists
Universitat Politècnica de Catalũnya (UPC)

Assignment 1, Design of a FE program

Simen Lieng - slieng8@gmail.com
Nicolas Andre Caronte Grønland - nagronla@stud.ntnu.no

Contents

1 Introduction

This assignment is the first part of a three-part assigment where the main goal is to code and beeing able to use a FE program made in MatLAB. The task in the first assigment is to produce an initial design of the FE program code. This does not include any direct form for MatLAB codes, and is more like a strategy and explanation of how we choose to solve the given problem.
The given problem is to make this initial design as flexible as possible. Therefor will the design consider different possibilities as stated:

- Different domains in both shape and order (1D, 2D and 3D).
- Different elements: 1D bar element. 2D quadrilateral and Triangle elements. 3D hexahedral and tetrahedral elements.
- Material properties depending on space location.

2 Methodology

For our program to solve the linear problem $\mathbf{Ku} = \mathbf{f}$, we chose to design one structure(green) and several functions(blue). Section 3 will show how these are linked together with each other.

2.1 Structuring the inputs

Structure name: FEinputs **Description:** organizes and saves the input parameters
.typeElem returns chosen element type: 1=Bar 2=Quadrilateral 3=Triagnle 4=Hexahedral 5=Tetrahedral
.orderElem returns the chosen order, 1 or 2
.BCs returns chosen boundary conditions Dirichlet: Prescribed u-values Neumann: Fluxfunction and boundary
.s returns $s(x,y)$ as function or value
.scalar returns μ as a scalar

2.2 Defining the mesh

Function name: defineMesh
Description: takes the chosen element type and order, defines a mesh and spits connectivity matrix and coordinates related to every node.
Inputs: `FEinputs.typeElem, FEinputs.orderElem`
Outputs: $\mathbf{T}, \mathbf{X}$
Uses: none **Used by:** none
Comments: The function will only be used once.

2.3 Defining the amount of integration points

The amount of integration points is dependent on which element is chosen, as well as the order of the element. We chose the Gauss Quadrature Rule to evaluate how many integration points we need for the given element. Gauss quadrature rules are designed to integrate exactly a polynomial of degree $2n-1$ by choosing n integration points (Gauss points).

Function name: quadrature
Description: takes the chosen element type and order, defines the needed amount of integration points and spits coordinates $\mathbf{Z_g}$ and weights w_g for reference element.
Inputs: `FEinputs.typeElem, FEinputs.orderElem`
Outputs: $\mathbf{Z_g}$, w_g
Uses: none **Used by:** none
Comments: size of both $\mathbf{Z_g}$ and w_g depend on the amount of integration points and will be general for all elements.

2.4 Defining the reference element

By using isoparametric mapping, every element in the established mesh could be described by using one reference element. This reference element is based on the element shape and order chosen as input.

Function name: referenceElement
Description: takes the chosen element type and order, and spits shape functions and their derivatives evaluated at $\mathbf{Z_g}$. Additionally it will spit **s** evaluated at $\mathbf{Z_g}$.
Inputs: `FEinputs.typeElem, FEinputs.orderElem, FEinputs.s` ,$\mathbf{Z_g}$
Outputs: N , dN , s
Uses: none **Used by:** none
Comments: the outputs are general for all elements.

2.5 Constructing K and f

When the reference element is established, the next step is to construct the K and f-matrices. This is done element by element in a for-loop containing the following functuons.

2.5.1 Define the Jacobian Matrix

Function name: elementJacobian
Description: takes derivatives of shape functions evaluated at $\mathbf{Z_g}$ together with the global coordinats of the nodes and spits inverse of jacobian and jacobian determinant for an element.
Inputs: $\mathbf{dN}$, $\mathbf{X}$
Outputs: $\mathbf{iJ}$, `det(J)`
Uses: none **Used by:** none
Comments: will be called once for every element(for-loop).

2.5.2 Establish the K^e matrix

Function name: elementK
Description: takes weights, inverse jacobian, derivatives of shape functions, scalar, the determinant of the jacobian and spits the K-matrix for each element.
Inputs: `FEinputs.scalar`, w_g, $\mathbf{iJ}$, `det(J)`, $\mathbf{dN}$
Outputs: $\mathbf{K^e}$
Uses: none **Used by:** none
Comments: will be called once for every element(for-loop).

2.5.3 Assemble K^e into global K matrix

Function name: AssembleK
Description: takes the K-matrix for each element and its corresponding connectivity matrix to assemble the system K-matrix.
Inputs: $\mathbf{K^e}$, $\mathbf{T}$
Outputs: $\mathbf{K}$
Uses: none **Used by:** none
Comments: will be called once for every element(for-loop).

2.5.4 Establish the f^e vector

Function name: elementF
Description: takes weights, inverse jacobian, the determinant of the jacobian, shape functions, s-function and spits the f-matrix for each element.
Inputs: `FEinputs.s`, w_g, $\mathbf{iJ}$, `det(J)`, $\mathbf{N}$
Outputs: $\mathbf{f^e}$
Uses: none **Used by:** none
Comments: will be called once for every element(for-loop).

2.5.5 Assemble f^e into global f vector

Function name: AssembleF
Description: takes the f-matrix for each element and its corresponding connectivity matrix to assemble the system f-matrix
Inputs: $\mathbf{f^e}$, **T**
Outputs: f
Uses: none **Used by:** none
Comments: will be called once for every element(for-loop).

2.6 Applying boundary conditions

This program can solve the linear system with either Dirichlet or Neumann boundary conditions. Corresponding functions will be used depending on the chosen method.

Function name: Dirichlet
Description: takes in the prescribed unknowns $u_i = \alpha$, the system K-matrix, and the system f-matrix. The column in the K-matrix corresponding to the prescribed unknowns will be deleted from the K-matrix and the outputs will be these columns multiplied by α. The rows corresponding to the prescribed unknowns will also be deleted from the K-matrix, f-matrix and unknowns.
Inputs: `FEinputs.BCs`, **f** , **K**
Outputs: Dirichlet, $\mathbf{K^{new}}$, $\mathbf{f^{new}}$
Uses: none **Used by:** none
Comments: Dirichlet is a vector and will be put on the right hand side of the linear system.

To use the Neumann boundary conditions the following two functions must be looped for every element on the Neumann boundary.

Function name: elementNeumann
Description: localizes a "new" mesh consisting of the edge of the elements that intercepts the Neumann boundary **q**. For every element along this boundary, there will be computed a integral along its intersection edge. The integral is solved using shape functions and a quadrature rule. This spits a vector containing the result of the integration along the element edge.
Inputs: q, **X**, **T**
Outputs: $\mathbf{Neu_{el}}$
Uses: none **Used by:** none
Comments: will be called once for every element along the Neumann boundary (for-loop).

Function name: AssembleNeumann
Description: takes the $\mathbf{Neu_{el}}$-vector for each element and its corresponding connectivity matrix to assemble the system **Neu**-vector.
Inputs: $\mathbf{Neu_{el}}$, **T**
Outputs: Neu
Uses: none **Used by:** none
Comments: will be called once for every element along the Neumann boundary (for-loop). **Neu** is a vector and will be put on the right hand side of the linear system.

2.7 Solving the linear system

When the global linear system is established, the system can be solved by using functions integrated in the MatLAB software. There will be two possible linear systems depending on the boundary conditions:

$$\mathbf{K^{new}u = f^{new} - Dirichlet} \tag{1}$$

$$\mathbf{Ku = f + Neu} \tag{2}$$

3 Dependency graph

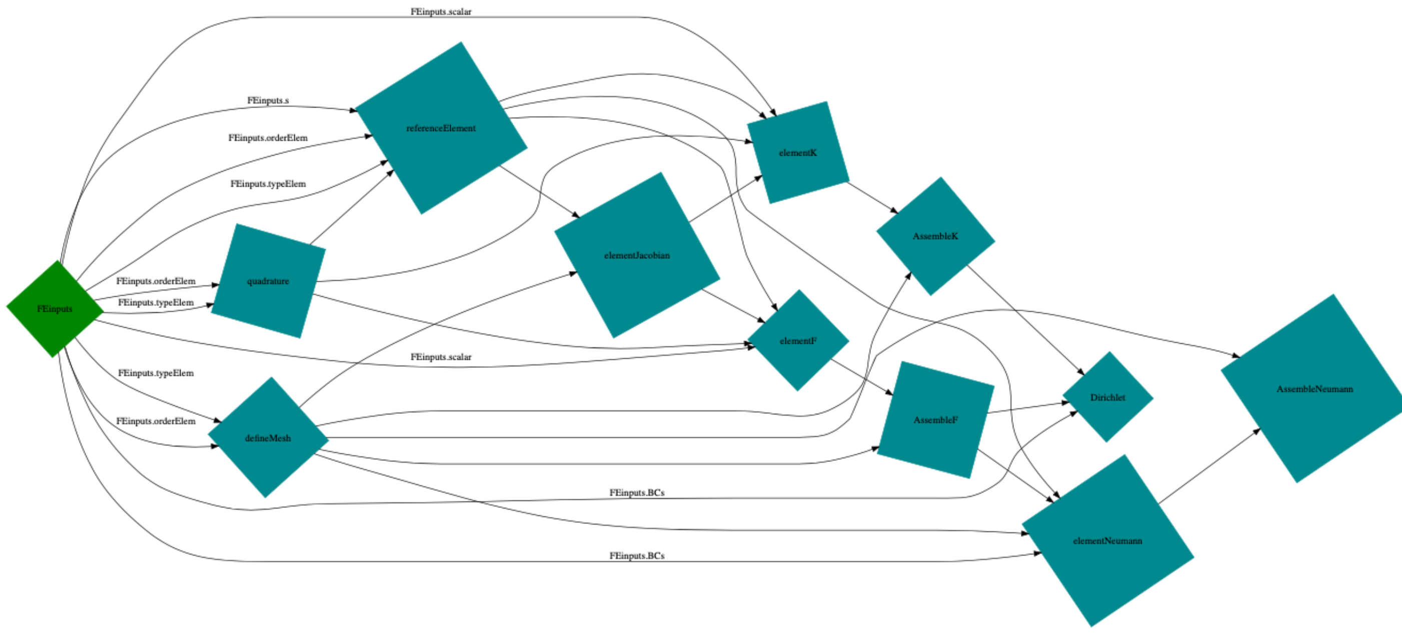

Figure 1: Illustration of the dependency between the functions and the structure.

4 Discussion

When assessing the solutions we've chosen, we think its reasonable to look at our lack of experience to rather advanced programming. Therefore we chose to divide the program into several smaller functions, to make the program easier to understand and less complicate to detect any errors. An alternative solution would have been to make larger functions with more code, roughly doing the same as a few of ours. This could have given a more compact and computationally cheaper code. The structure used is only to save input variables. This is the first time using this kind of data type, and were therefore not used more than once.

Furthermore, the functions regarding the boundary conditions are prone to some modification. Our experience with finite element method is limited and we are therefore not completely sure how to implement these exactly. This is especially true for the Neumann-part and how to implement or mesh the respective boundary.

Lastly, the functions are each described by a similar box as in the example from the first MatLAB-presentation. A part of the exercise was to identify if a given function uses some of the functions in the program. To clarify why none of our functions are either used by or uses other functions, is because none of them calls directly upon another given function. This does not mean there is no dependency of the functions, because quite a few uses the actual output from other functions. The output is saved directly in the MatLAB-program, and therefore makes it unnecessary to call upon the other functions. The actual dependency of the functions and the structure can be shown in the dependency graph.

5 Sources

The only sources used for this assignment is the two powerpoints about MatLAB found in Cimne.