# ****Programming Project Report****

## ****1.0 Overview****

The goal of this project was to develop a program capable of solving one-dimensional boundary value problems (BVPs) using the finite element method (FEM). The program was designed to handle various boundary conditions, including essential (EBC), natural (NBC), and mixed (MBC) boundary conditions. The program was developed in GUN C++ compiler based on Linux system.

### ****1.1 General Requirements****

The program was developed with the following requirements in mind:

1. **Formatted Input:** The input values are based on formatted input text file.
2. **Formatted Output:** The output result are stored in text file, and display on the command windows.
3. **Language**: The program was written in ANSI C++. Initially developed in LINUX environments and fixed the problem running on the windows system.
4. **Units**: The program assumes consistent units for all input data.
5. **Debugging**: A debugging variable was implemented to control the level of debugging output in the output file.

## ****2.0 Program Implementation****

### ****2.1 Input File Format****

The input file is divided into several sections, each containing specific data required for the finite element analysis. The sections are as follows:

**Section 1**: Problem identification and description.

**Section 2**: Alpha values (only support constant).

**Section 3**: Beta values (only support constant).

**Section 4**: Force values (only support constant).

**Section 5**: Nodal coordinates.

**Section 6**: Nodal flux values.

**Section 7**: Element data, including element type and connectivity.

**Section 8**: Left end boundary condition (EBC, NBC and MBC).

**Section 9**: Right end boundary condition (EBC, NBC and MBC).

**Section 10**: End of input file.

The program reads the input file, parses the data, and stores it in appropriate data structures.

### ****2.2 Solution building****

#### 2.2.1 Element Stiffness Matrix and Element Stiffness Load

**step 1**. Reading the input file and storing the data to program.

For alpha, beta and force. Record the index integer and double variables, to find the value later. The values are stored in a vector with a self-costume type. For Element data, store the integer for which to use and type of element to apply and store the node index in a vector. And boundary condition, store the c and d value, and use them with boundary condition. Details can check for Process.h.

**Step 2**: Building the System Stiffness Matrix (K) and System Load Vector (F)

Once the input data is stored, the program proceeds to build the **System Stiffness Matrix (K)** and the **System Load Vector (F)** based on the element types specified in the input file. The program supports two types of elements:

1. **1D Linear Element (1DC0L)**
2. **1D Quadratic Element (1DC0Q)**

The construction of **K** and **F** depends on the type of element used in the model. Below is a detailed explanation of how **K** and **F** are built for each element type.

##### 2.2.1.1 1D Linear Element (1DC0L)

For a **1D Linear Element**, the **Element Stiffness Matrix (ESM)** and **Element Load Vector (ELV)** are constructed using the following formulas:

**Element Stiffness Matrix (ESM):**

**Where:**

**A is the alpha value for the element.**

**B is the beta value for the element.**

**L is the length of the element**

**Element Load Vector (ELV):**

Where:

*f* is the force value for the element.

##### 2.2.1.2 1D Quadratic Element (1DC0Q)

For a **1D Quadratic Element**, the **Element Stiffness Matrix (ESM)** and **Element Load Vector (ELV)** are constructed using the following formulas:

**Element Stiffness Matrix (ESM)**:

**Where:**

**A is the alpha value for the element.**

**B is the beta value for the element.**

**L is the length of the element**

**Element Load Vector (ELV):**

Where:

*f* is the force value for the element.

#### 2.2.2 System Stiffness Matrix and System Stiffness Load

The size of the matrix depends on the number of nodes in the model. For a 1D problem with n nodes, the matrix will be of size n×n. For construct matrix K, basing on each element information do step 2.2.1calculates the **Element Stiffness Matrix (ESM)** based on the element type (linear or quadratic). For a 1D linear element, it is a 2×2 matrix, and for a 1D quadratic element, it is a 3×3 matrix. Placing the value to their node number in the matrix K. Each element's ESM is added to the appropriate positions in the global matrix based on the connectivity of the nodes. For example, If we have element 1 and element 2 for whole system with linear mesh, then element 1 will added to and element 2 will added to . This process is repeated for all elements, and the contributions are summed to form the complete **System Stiffness Matrix (K)**.

**For a 1D linear element, the ELV is a 2×1 vector, and for a 1D quadratic element, it is a 3×1 vector. Each element's ELV is added to the appropriate positions in the global vector based on the connectivity of the nodes. Similar, to System Stiffness Matrix (K)**, If we have element 1 and element 2 for whole system with linear mesh, add all load on node 1 to , add all load on node 2 to and add all load on node 3 to . This process is repeated for all elements, and the contributions are summed to form the complete **System Load Vector (F)**.

#### 2.2.3 Boundary Conditions and Modification of K and F

After assembling the global stiffness matrix (K) and force vector (F), the program applies the boundary conditions specified in the input file. These boundary conditions can be classified into three types: Essential Boundary Conditions (EBC), Natural Boundary Conditions (NBC), and Mixed Boundary Conditions (MBC). Essential Boundary Conditions enforce specified displacement values at boundary nodes by modifying the K and F matrices. For instance, if a node has a displacement of zero, the program removes the corresponding row and column from K and adjusts F accordingly. Natural Boundary Conditions, on the other hand, account for specified flux values at boundary nodes by modifying only the F vector. If a node has a flux value q, this value is added to the corresponding entry in F. Lastly, Mixed Boundary Conditions involve a combination of essential and natural conditions, requiring modifications to both K and F. If a node is subject to an MBC with coefficients c and d, the program adjusts the respective entries in K and F to reflect these conditions. Through these modifications, the program ensures that the system of equations accurately represents the problem's physical constraints.

#### 2.2.3.1 Essential Boundary Conditions

For left end boundary condition is of type **EBC**, the program subtracts the product of the first column of **K** and the specified displacement value *c* from the load vector **F**. Then program removes the first row and first column from K and the first entry from F to eliminate the fixed node from the system of equations. For right end boundary condition is of type **EBC**, the program subtracts the product of the last column of **K** and the specified displacement value *c* from the load vector **F**. Then program removes the last row and last column from K and the last entry from F to eliminate the fixed node from the system of equations.

#### 2.2.3.2 Natural Boundary Conditions

If the left end boundary condition is of type **NBC**, the program adds the specified flux value d*d* to the first entry of the load vector **F**. If the right end boundary condition is of type **NBC**, the program subtracts the specified flux value d*d* from the last entry of the load vector **F.**

#### 2.2.3.3 Mixed Boundary Conditions

If the left end boundary condition is **MBC**, the program modifies both **K** and **F** to account for the mixed condition. If the right end boundary condition is **MBC**, the program modifies both **K** and **F** to account for the mixed condition.

**2.3 Solve unknown**

the system of equations K⋅D=F is solved using **Gaussian elimination with partial pivoting**. This method is implemented in two main steps: **Forward Elimination** and **Backward Substitution**. These steps are performed in the Forward() and Backward() functions within the Process class. This method is implemented in the Solve() function, which takes the **System Stiffness Matrix (K)** and the **System Load Vector (F)** as inputs and returns the solution vector D.

**2.4 Getting Flux**

#### 2.4.1 Flux Calculation for 1D Linear Elements (1DC0L)

For each Element and solution valuables that solved, the flux is calculated at the midpoint of the element using the following formula:

Where:

A is the alpha value for the element (material property).

are the nodal displacements at the two nodes of the element.

L is the length of the element, calculated as the difference between the nodal coordinates of the two nodes.

And Flux position, is defined as:

#### 2.4.2 Flux Calculation for 1D Quadratic Elements (1DC0Q)

For each Element and solution valuables that solved, the flux is calculated at the midpoint of the element using the following formula:

Where:

A is the alpha value for the element (material property).

are the nodal displacements at the three nodes of the element from left, medial, right.

L is the length of the element, calculated as the difference between the coordinates of the first and last nodes.

And Flux position, is defined as:

If the quadratic displays are the same which means the difference between three are less than 1e-10. The flux will display 0.

**2.5 Regrade Debug**

The difference in line endings between Linux and Windows systems. In Linux, lines in text files typically end with a newline character (\n), while in Windows, lines end with a carriage return followed by a newline (\r\n). This difference can cause problems when reading files in C++ programs, especially when comparing strings or parsing input. In the program code, the problem arises when checking the currentSection string. On Windows, the currentSection string may include an extra \r character at the end of the line, which causes the comparison to fail.

Replace

if (currentSection == "alpha")

To

if (currentSection == "alpha" || currentSection == "alpha\r")

without reading “\r” in windows, the input file parse program will have nothing for the input, even though the matrix load vector still can be built. But it may create an empty matrix and empty vector.

Fixed the quadratic flux displacement formula and flux value in the code. Details check in section 2.4.

### 2.6 Error analysis

The program computes both the absolute and relative error norms for the solution. The absolute error is given by , where R=KD−F, and the relative error is given by . These error measures provide insight into the accuracy of the solution.

## ****3.0 Running program****

In visual studio code command Enter:

g++ -std=c++17 -o main main.cpp Process.cpp

then run main.exe in the command windows. System will ask the debug level for the coding process. Enter 0, will just print all the output, enter 1 will print all the system stiffness matrix and system load vector, and enter 2 will print all the element stiffness matrix, element load vector, system stiffness matrix, and system load vector.

Enter the name of the input file and name of the output file.

### ****3.1 Test Model 1****

Input file name = Test Model 1.txt

Output file name = Test Model 1.out

result shows on Test Model 1.out in the zip file

### ****3.2 Test Model 2****

Input file name = Test Model 2.txt

Output file name = Test Model 2.out

result shows on Test Model 2.out in the zip file

### ****3.3 Test Model 3****

Input file name = Test Model 3.txt

Output file name = Test Model 3.out

result shows on Test Model 3.out in the zip file

### ****3.4 FEFE\_TP\_1****

Input file name = **FEFE\_TP\_1**.txt

Output file name = **FEFE\_TP\_1**.out

result shows on **FEFE\_TP\_1**.out in the zip file

### ****3.5 FEFE\_TP\_2****

Input file name = **FEFE\_TP\_2**.txt

Output file name = **FEFE\_TP\_2**.out

result shows on **FEFE\_TP\_2**.out in the zip file

### ****3.6 FEFE\_TP\_3****

Input file name = **FEFE\_TP\_3**.txt

Output file name = **FEFE\_TP\_3**.out

result shows on **FEFE\_TP\_3**.out in the zip file

### ****3.7 FEFE\_TP\_4****

Input file name = **FEFE\_TP\_4**.txt

Output file name = **FEFE\_TP\_4**.out

result shows on **FEFE\_TP\_4**.out in the zip file

### ****3.8 FEFE\_TP\_5****

Input file name = **FEFE\_TP\_5**.txt

Output file name = **FEFE\_TP\_5**.out

result shows on **FEFE\_TP\_5**.out in the zip file

### ****3.9 FEFE\_TP\_6****

Input file name = **FEFE\_TP\_6**.txt

Output file name = **FEFE\_TP\_6**.out

result shows on **FEFE\_TP\_6**.out in the zip file