Development of a Graphic Program for Structural Analysis of Linear Element Models

Rafael Lopez Rangel

Department of Civil Engineering, Pontifical Catholic University of Rio de Janeiro – PUC-Rio

Emphasis: Structures

Area: Computational Mechanics

Supervisor: Luiz Fernando Campos Ramos Martha

Rio de Janeiro, December 2016

For Anna and Paulo

List of Figures

Figure 1.1 Comparison of analysis results using different analysis types 4

Figure 1.2 Four levels of abstraction of a structural system (Martha, 2010).....5

Figure 2.1 Different types of linear element models (analysis models): plane truss, spatial truss, plane frame, spatial frame and grillage.....8

Figure 2.2 Local axes convention of any element in LESM.....10

Figure 2.3 Positive orientations of internal forces in plane truss models 13

Figure 2.4 Degrees of freedom numbering order of a plane truss element.....13

Figure 2.5 Structural model adopted to idealize the behavior of a trussed bridge.....14

Figure 2.6 Positive orientations of internal forces in plane frame models 15

Figure 2.7 Degrees of freedom numbering order of a plane frame element.....16

Figure 2.8 Structural model adopted to idealize the behavior of a portal frame 17

Figure 2.9 Positive orientations of internal forces in grillage models 18

Figure 2.10 Degrees of freedom numbering order of a grillage element.....19

Figure 2.11 Structural model adopted to idealize the transversal behavior of beams .19

Figure 3.1 Influence of shear deformation according to the length of the elements ...22

Figure 4.1 Flowchart of the operating stages of LESM.....31

Figure 5.1 Flowchart of the classes of LESM.....40

Figure 5.2 Properties and methods of the *Drv* class 41

Figure 5.3 Properties and methods of the *Anm* class 42

Figure 5.4 Properties and methods of the *Elem* class 44

Figure 5.5 Properties and methods of the *Lelem* class.....46

Figure 5.6 Properties and methods of the *Node* class 47

Figure 5.7 Properties and methods of the *Material* class.....47

Figure 5.8 Properties and methods of the *Section* class.....48

Figure 5.9 Properties and methods of the *Draw* class 49

Figure 5.10 Properties and methods of the *Print* class 50

Figure 5.11 Main window of the graphical user interface of LESM 51

Figure 5.12 Auxiliary GUI for managing material objects 54

Figure 5.13 Auxiliary GUI for managing cross-section objects 54

Figure 5.14 Auxiliary GUI for managing node objects 55

Figure 5.15 Auxiliary GUI for managing element objects 55

Figure 5.16 Auxiliary GUI for managing support conditions and prescribed displacements 56

Figure 5.17 Auxiliary GUI for managing nodal loads 57

Figure 5.18 Auxiliary GUI for managing distributed loads 57

Figure 5.19 Settable properties of each analysis model 60

Figure 5.20 Flowchart of the methods to assemble the equation system 63

Figure 5.21 Flowchart of the methods to compute internal forces at element ends 70

Figure 5.22 Panel of results options 72

Figure 5.23 Lower face of elements in plane truss and plane frame models 72

Figure B.1 Fixed end forces from axial distributed load in the XY plane 84

Figure B.2 Fixed end forces from axial distributed load in the XZ plane 84

Figure B.3 Fixed end forces from transversal distributed load in the XY plane 85

Figure B.4 Fixed end forces from transversal distributed load in the XZ plane 85

Figure C.1 Fixed end forces from axial temperature variation in the XY plane 89

Figure C.2 Fixed end forces from axial temperature variation in the XZ plane 89

Figure C.3 Fixed end forces from temperature gradient in the XY plane 90

Figure C.4 Fixed end forces from temperature gradient in the XZ plane 90

Figure E.1 Element loaded with distributed axial load 98

Figure E.2 Element loaded with distributed transversal load 98

Figure F.1 Element with temperature variation 101

Contents

1 Introduction 1

1.1 Overview 1

1.2 Objectives 6

1.3 Document Organization 7

2 Linear Element Models 8

2.1 Overview 8

2.2 General Considerations 9

2.3 Plane Truss 12

2.4 Plane Frame 15

2.5 Grillage 17

3 Mathematical Formulations for Linear Elements 20

3.1 Overview 20

3.2 Beam Theories 20

3.3 Stiffness Matrices 23

3.4 Fixed End Forces 25

3.5 Shape Functions 27

3.6 Internal Displacements in Loaded Elements....28

4 Computational Implementation....29

4.1 Overview....29

4.2 Implementation stages of a typical graphic program of structural analysis....29

4.3 The Use of MATLAB to Develop a Graphic Program....32

4.4 Object-Oriented Programming Paradigm....34

4.5 Event-Driven Programming Paradigm....38

5 The Development of LESM....39

5.1 Overview....39

5.2 The Classes of LESM....39

5.2.1 The *Drv* (Driver) Class....41

5.2.2 The *Anm* (Analysis Model) Class....42

5.2.3 The *Elem* (Element) Class....42

5.2.4 The *Lelem* (Load Element) Class....45

5.2.5 The *Node* Class....46

5.2.6 The *Material* Class....47

5.2.7 The *Section* Class....47

5.2.8 The *Draw* Class....48

5.2.9 The *Print* Class....49

5.3 The Graphical User Interface....50

5.4 Data Pre-Processing....59

5.5 Data Processing....61

5.5.1 Assembly of the Equation System....62

5.5.2 Resolution of the Equation System....68

5.5.3 Computation of Internal Forces at Element Ends....69

5.6 Data Post-Processing....71

5.6.1 Deformed Configuration....73

5.6.2 Axial Force Diagram....75

5.6.3 Shear Force Diagram....76

5.6.4 Bending Moment Diagram....77

5.6.5 Torsion Moment Diagram....77

5.6.6 Other Result Options....78

6 Conclusions....79

References....80

Appendix A Local Stiffness Coefficients of Linear Elements.....81

Appendix B Fixed End Forces from Linearly Distributed Load 84

Appendix C Fixed End Forces from Temperature Variation.....89

Appendix D Shape Functions 93

Appendix E Internal Displacements from Linearly Distributed Load.....98

Appendix F Internal Displacements from Temperature Variation 101

1 Introduction

1.1 Overview

Structural engineers and architects, in order to design buildings, bridges, factories, and any other type of structural system, they need to predict how the structure components, such as beams, columns and slabs, and also the system as a whole, will behave under the effect of all possible loads and unexpected displacements that can occur over the structure life cycle. This is done by calculating internal forces, stresses and deformations on the structure components. The field of study responsible for evaluating the effects of loads and displacements on structures is called structure analysis and must be mastered by all the professionals who work with the design, construction and maintenance of structural systems.

To apply the structural analysis methods it is necessary to represent the real structure as a structural model. This model is an idealization of the behavior of the real structure, so its components can be mathematically modeled by formulations developed in the solid mechanics theories. The conception of a structural model involves the adoption of simplifying assumptions about the model geometry, the material behavior, the support conditions with the external environment and how the loads are being applied to the structure.

The geometric simplifications are made to represent the structure components with complex geometry into simpler elements, called structural elements. These elements can be linear, surface or volumetric. Linear elements are the idealization of components with one dimension much greater than the other two, so that it can be represented by a line on its longitudinal axis. Examples of linear elements are beams

and columns. Surface elements represent components with two dimensions much greater than the other, such as slabs and shells. Volumetric elements are the representation of components without a prevailing dimension, such as pad foundations.

The materials used in the structure components need to have their behavior described by a constitutive law that relates the stresses and strains on elements. It is also necessary to determine if the material is homogeneous or heterogeneous, a characteristic related to the uniformity of its mechanical properties at different points of the element, and if it is isotropic or anisotropic, a characteristic related to the uniformity of its mechanical properties when measured in different directions.

The connection between the structure and the external environment, like other structures or the soil, must also be simplified with support conditions. The supports are responsible for allowing, or not, the structural model to move in a certain direction. These movements can be translations and rotations in the direction of an orthogonal axis. Plane models can have three possible movements (translation on horizontal and vertical axes and rotation on the same plane of the model), while spatial models can have six possible movements (translation on the three orthogonal axes and rotation on the three orthogonal planes). A support can restrain totally or partially one of these movements at the end of a structural element. A total restraint imposes the condition of null displacement in a certain direction, and a partial restraint allows a displacement that depends on the stiffness of the support. Most of the time, this simplification does not depend only on the structural model itself but also on the behavior of the supporting structures. For example, a column can be considered completely fixed to the soil, by its foundation, or there can be partial rotation liberation depending on the soil rigidity.

The internal connections between elements can also be modeled according to the transmission of the internal forces. Internal forces are the integral of the stresses in a cross-section that can generate a normal force, shear force, bending moment and torsion moment.

Another simplification that must be done is how the loads are applied to the model. Loads can act punctually on an element or distributed over a line or an area. Point loads can be considered when the contact area between the load agent and the element is so small that the effects of load distribution are negligible. An example of this is the case of a person standing in the middle of a slab. Distributed loads are considered when the effects of the load distribution in one or two dimensions cannot be ignored, such as the support of a slab on a beam, or the snow accumulation on a roof. The value of a distributed load follows a function that is usually constant or linear to simplify what happens on reality that can be very difficult to model, such as the value of the wind load in a building facade. Loads can also be static or dynamic, and this must be taken into account when conceiving the structural model.

The task of conceiving a structural model by assuming all these mentioned simplifications and deciding what considerations better represent the behavior of the real structure is called modeling. This task is a very important part of the job of engineers and architects because assuming different considerations can produce big changes in the analysis results, since each type of structural element, under the effect of a certain solicitation and support condition, combined with the constitutive law adopted for the materials, has different mathematical formulations to describe its behavior.

The results may also change depending on the analysis type that can consider second order effects and the plastic behavior of the material. In a first-order analysis, it is assumed the hypothesis of small displacements, which means that the resulting displacements of the structure are very small compared to its dimensions. Because of this, the internal forces are evaluated considering the undeformed configuration of the structure. These assumptions make the deformations and the internal forces to be proportional to the applied loads, and as a consequence the principle of superposition of effects can be used to simplify the analysis. In a second-order analysis the deformation of the structure is considered in the evaluation of the internal forces. As a result, the deformations and the internal forces are not proportional to the applied loads (geometric nonlinear effect). The analysis type also considers the constitutive law of the materials by assuming that they always behave elastically (elastic analysis), what makes the stresses to be linearly related to the strains, or considering their plastic behavior wherein stresses and strains cease to be proportional to a certain stage of solicitation. Figure 1.1 gives an idea of how the adoption of different analysis types influences the results of the same structural model.

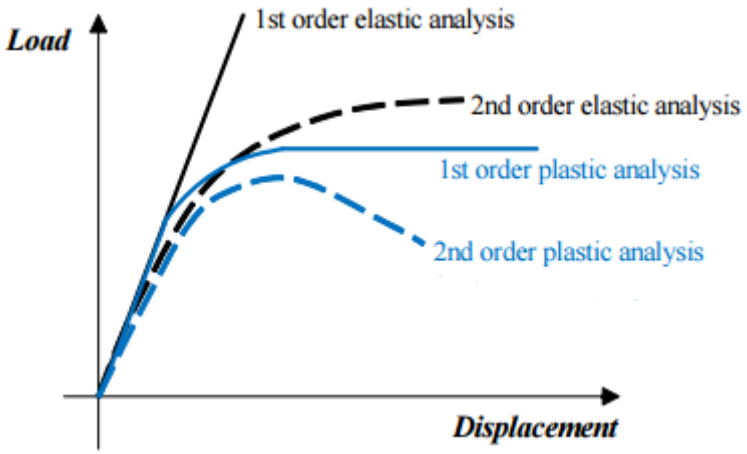

Figure 1.1 Comparison of analysis results using different analysis types

These changes in the results, caused by different modeling considerations and analysis types, can lead to a project that does not meet the security needs or is uneconomical.

Many methods can be used to analyze a structural model and obtain the desired results, but for this, the model must be discretized in parameters that are used to substitute the continuous analytical solution by the discrete values of these parameters. The adopted parameters, used in the discrete model, can be forces (loads or moments) or displacements, depending on which method is being used. The problem with these methods is that they require manual calculations that can turn this task to be impracticable because of the time it would take and the probability of errors.

With the advent of computers, a new idealization of the structural system became possible: the computational model. This turned the structural analysis into a process of computational simulation of structures behavior and allowed the implementation of analysis methods that would be impracticable to be done manually even for simple structural systems, such as the direct stiffness method and the finite element method. This also made it possible for users of structural analysis programs to visualize the analysis results on the screen of the computer. Nowadays it is unthinkable to design structures without a computer program.

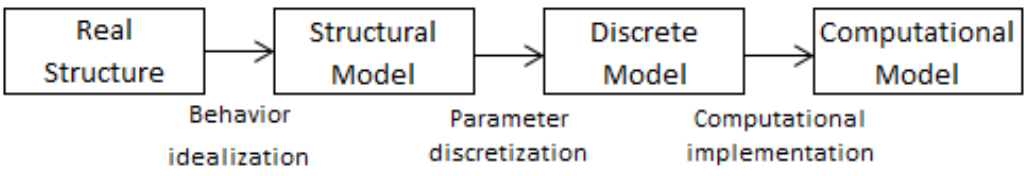

Figure 1.2 Four levels of abstraction of a structural system (Martha, 2010)

1.2 Objectives

The goal of this work is to show the steps of developing a structural analysis graphic program according to the direct stiffness method and provide the fundamental knowledge about the two major areas involved in this process: structural analysis and computer science.

To meet the objectives of this work, a graphic program for linear-elastic structural analysis of linear element models (LESM – Linear Elements Structure Model) was developed using the MATLAB programming language. This program currently has two versions, a graphical version, on which this work is based, and a non-graphical version for educational purposes. The documentation and the commented code of the non-graphical version can be found at http://webserver2.tecgraf.puc-rio.br/~lfm/analestrut3-162/lesm/main.html.

The LESM program considers only linear element models with linear elastic behavior (first-order elastic analysis) submitted to static loads. The algorithm of its data processing, which is the code responsible for calculating the structural analysis results, is based on the direct stiffness method. There are three types of structural models that can be analyzed: plane truss, plane frame and grillage.

The characteristics of LESM described throughout this work may be outdated due to the fact that the program is constantly evolving.

1.3 Document Organization

This work is divided in five chapters. Chapter 2 (Linear Element Models) describes the assumptions and idealizations of the three types of structural models considered in LESM.

Chapter 3 (Mathematical Formulations for Linear Elements) is dedicated to present the two theories that are used to describe the behavior of linear elements: Euler-Bernoulli (or Navier) beam theory and Timoshenko beam theory. This chapter also provides the mathematical formulations, in a matrix form, necessary for implementing the direct stiffness method.

Chapter 4 (Computational Implementation) focuses on explaining some concepts of computer science necessary for the development of LESM or any other structural analysis program. It also justifies the use of MATLAB and brings the main characteristics of the object oriented and event-driven programming paradigms.

Chapter 5 (The Development of LESM) is the main chapter of this work because it gives a full description of the LESM program, combining the concepts of structural analysis and computer science presented in the previous chapters. All the developed classes of the object-oriented programming paradigm are presented, as well as the functionalities of the graphical user interface. It also distinguishes the stages of data processing, pre-processing and post-processing by explaining their considerations and commenting how they are implemented on the code. The structure of the code is given in the form of flowcharts to help its understanding.

2 Linear Element Models

2.1 Overview

Linear element models are structural models composed only of elements that represent the geometric idealization of a structure component with its length much greater than the dimensions of its cross-section. This idealization consists in representing the component by a thin line that follows its longitudinal axis, and adopting particular mathematical formulations to describe its behavior.

Linear element models can be classified according to the model geometry, the type of connectivity between elements, and the direction of the applied loads. The most common types of linear element models are plane truss, spatial truss, plane frame, spatial frame and grillage, each one having its own considerations and assumptions. Different types of structural models are also called analysis models because of the distinct treatment they receive in the process of structural analysis. In the following sections it will be described the characteristics of the analysis models considered in LESM, which are the plane truss, plane frame and grillage models.

Figure 2.1 Different types of linear element models (analysis models): plane truss, spatial truss, plane frame, spatial frame and grillage

2.2 General Considerations

In linear element models, an element can be rectilinear or curved, with constant or variable cross-section, and it is defined by two nodal points: initial node and final node. Nodes that allow relative rotation of elements connected to them are called hinged, while those that do not allow relative rotation are called rigid. The angle formed by elements interconnected by rigid nodes is the same before and after the structure deforms, which may require the action of a bending moment to preserve the original angle. In hinged nodes, the relative rotation between elements causes the angle in the deformed configuration to be different from the originally set in the undeformed configuration, and the bending moment at this point to be zero. In LESM, all models are composed by rectilinear elements with constant cross-section and fixed or hinged connections at their ends.

The models are considered to be laid in the XY plane and there are two coordinate systems to reference them: the global coordinate system, based on the global axes, and the local coordinate system, based on the local axes of elements. The global axes specify a direction or the coordinates of a point relative to a fixed reference. The local axes of an element, in the plane models of LESM, are defined uniquely for all types of models in the following manner:

- The local axis Z of an element is always in the direction of the global axis Z, which is perpendicular to the plane of the model.
- The local axis X of an element follows its longitudinal axis from the initial node to the final node.
- The local axis Y of an element is perpendicular to the axis X in such a way that the cross-product results in a vector in the global Z direction.

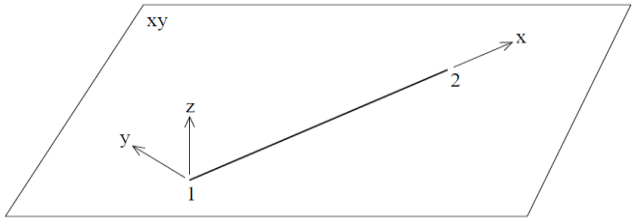

Figure 2.2 Local axes convention of any element in LESM

The degrees of freedom of an element are the combination of the degrees of freedom of its initial and final nodes, which are the possible directions that a node can move in a certain analysis model. These directions are identified by numbers and include translations and rotations that can be expressed in the global or local system. The numbering order of the degrees of freedom of an element, in the local system, is presented in the next sections for each analysis model type.

In models discretized by nodal displacements, which is the case of the direct stiffness method, the deformed configuration of an element is based on the displacements of its initial and final nodes. A nodal displacement in this context is a movement component in the direction of an element degree of freedom. These displacements are assembled in vectors:

- {d} → Vector of the nodal displacements of an element in the global system.
- {d'} → Vector of the nodal displacements of an element in the local system.

The internal forces of an element are based on its generalized forces, which are loads and moments acting in the direction of an element degree of freedom to equilibrate it in a deformed configuration. These forces are also assembled in vectors:

- {f} → Vector of the generalized forces of an element in the global system.
- {f'} → Vector of the generalized forces of an element in the local system.

It is often necessary to convert these displacements and forces vectors from one coordinate system to another, so that a rotation transformation matrix is required. This matrix is calculated from geometric relations using the inclination angle of the elements, and the result is an orthogonal matrix (its transpose is equal to its inverse) that is valid for any element inclination and any numbering order for the element end nodes. Each analysis model has a particular rotation transformation matrix, which are presented in the next sections. Equations 2.1 and 2.2 demonstrate how to convert the vector of nodal displacements from one coordinate system to another, where [R] is the element rotation transformation matrix. The conversion of the vectors of generalized forces is done exactly the same way.

$$\{d'\} = [R]\{d\} \tag{2.1}$$

$$\{d\} = [R]\{d'\} \tag{2.2}$$

There are four types of loads considered in LESM and it is assumed that there is a single load case. The load types are:

- Concentrated nodal load in global axis directions.
- Uniformly distributed load on elements, spanning its entire length, in local or in global system.
- Linearly distributed load on elements, spanning its entire length, in local or in global system.
- Uniform temperature variation on faces of elements, resulting on axial uniform temperature variation and uniform temperature gradient.

In addition, nodal prescribed displacements may be specified, always on global system.

2.3 Plane Truss

A plane truss model is a common form of analysis model because of its simplicity. The following assumptions are adopted for this model type:

- All of its elements and loads belong to the XY plane.
- Truss elements are connected at their ends by frictionless pins (hinges). Therefore, a truss element does not present any secondary bending moment induced by rotation continuity at joints.
- A truss model is loaded only at joints (nodal points or just nodes). Any load action along an element, such as self-weight, is statically transferred as concentrated forces to the element end nodes.

- In LESM, this model accepts only loads (concentrated nodal loads or distributed loads on elements), not moments. The components of concentrated nodal loads are always specified in the global system, while the components of uniformly or linearly distributed loads on elements can be specified in the global or local systems.

- Local bending of elements due to transversal loads is neglected, when compared to the effect of global load acting on the truss.
- There is only one type of internal force in a plane truss element: axial internal force, which may be tension or compression. The convention for the positive orientation of internal forces in plane truss models is represented in Figure 2.3.

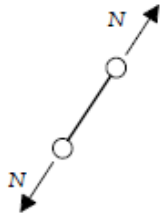

Figure 2.3 Positive orientations of internal forces in plane truss models

- As truss elements have, by assumption, only axial behavior, the area of its cross-section is the single geometric parameter needed for the analysis.
- Each node of a plane truss model has two degrees of freedom: horizontal and vertical translation components. The degrees of freedom numbering order considered in LESM for a truss element is shown in Figure 2.4 in the positive directions of the local system, where β is the element angle with the global X direction.

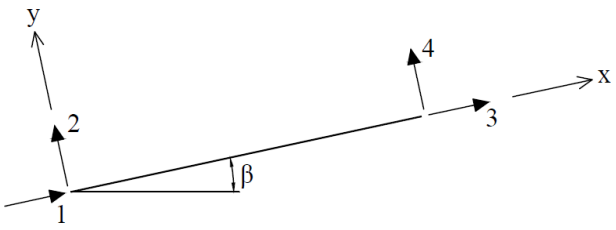

Figure 2.4 Degrees of freedom numbering order of a plane truss element

- The rotation transformation matrix of a plane truss element is:

$$[\text{rot}] = \begin{bmatrix} \cos(\beta) & \sin(\beta) & 0 & 0 \\ -\sin(\beta) & \cos(\beta) & 0 & 0 \\ 0 & 0 & \cos(\beta) & \sin(\beta) \\ 0 & 0 & -\sin(\beta) & \cos(\beta) \end{bmatrix} \tag{2.3}$$

- Supports of the first and second genders are considered i.e., total restriction of horizontal and/or vertical translation.
- As there are no rotation restrictions on the supports, this model accepts only translational (horizontal or vertical) prescribed displacements in the direction of fixed degrees of freedom.

Considering the previous description of a truss analysis model, there are some rules that must be followed when conceiving this type of structural model, called the truss basic formation law, necessary for maintaining the stability of the model. It stablishes that if a new node is added to an isostatic plane truss, this node must be connected to two other nodes by inserting elements between them. This is required because a new node corresponds to the addition of two degrees of freedom, or two unknown variables on the system of equilibrium equations, and each element corresponds to the addition of a new equation to the system. The result of this is that the minimum unit to create a plane truss model is a triangle. Therefore, this model type is usually formed by a triangulation between its hinged elements.

Figure 2.5 exemplifies the conception of a possible structural model of a trussed bridge, for the analysis of the plane behavior of one side of the bridge.

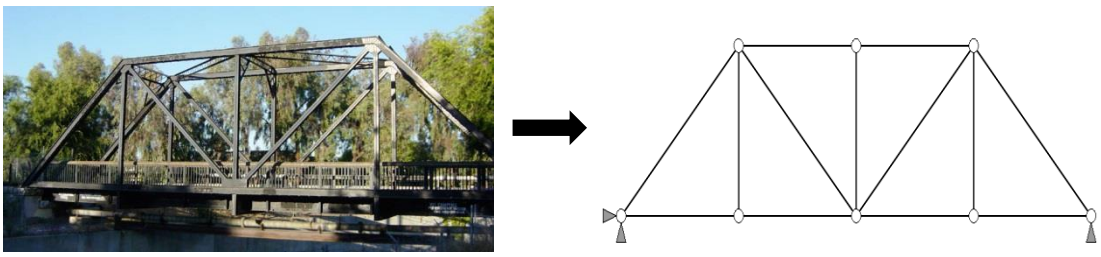

Figure 2.5 Structural model adopted to idealize the behavior of a trussed bridge

2.4 Plane Frame

A frame model is made up of beams (horizontal and inclined elements) or columns (vertical elements). A continuous beam (an assemblage of connected beams) is considered as a frame model by LESM. The assumptions of a plane frame model are:

- All of its elements and loads belong to the XY plane.
- By assumption, there is only in-plane behavior, which means that there is no displacement transversal to the plane of the model.
- Frame elements are usually rigidly connected at the joints. However, a frame element might have a hinge (rotation liberation) at an end or hinges at both ends. A frame element with hinges at both ends works like a truss element.
- In LESM this model accepts concentrated nodal loads and moments, and distributed loads on elements. The components of concentrated nodal loads and moments are always specified in the global system, while the components of uniformly or linearly distributed loads on elements can be specified in the global or local systems.
- Internal forces at any cross-section of a plane frame element are: axial force, shear force, and bending moment. The convention for the positive orientation of internal forces in plane frame models is represented in Figure 2.6.

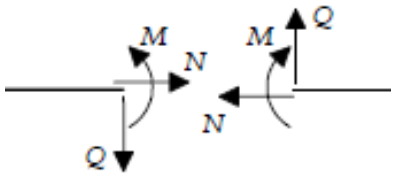

Figure 2.6 Positive orientations of internal forces in plane frame models

- The geometric parameters needed for the analysis of a plane frame model are the full area, the effective shear area, and the bending inertia of the cross-sections.
- Each node of a plane frame model has three degrees of freedom: horizontal and vertical translation components and in-plane rotation. The degrees of freedom numbering order considered in LESM for a plane frame element is shown in Figure 2.7 in the positive directions of the local system, where β is the element angle with the global X direction.

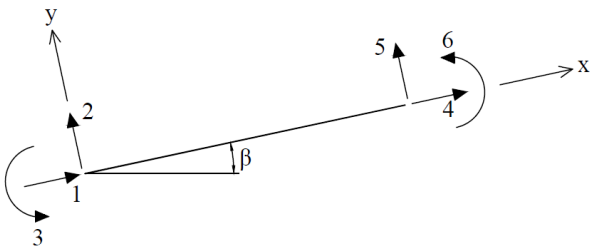

Figure 2.7 Degrees of freedom numbering order of a plane frame element

- The rotation transformation matrix of a plane frame element is:

$$[\text{rot}] = \begin{bmatrix} \cos(\beta) & \sin(\beta) & 0 & 0 & 0 & 0 \\ -\sin(\beta) & \cos(\beta) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos(\beta) & \sin(\beta) & 0 \\ 0 & 0 & 0 & -\sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

- Supports of the first, second and third genders are considered i.e., total restriction of horizontal or vertical translation, and to node rotation.
- This model accepts both translational (horizontal or vertical) and rotational prescribed displacements in the direction of fixed degrees of freedom.

Plane frame models are normally used to analyze the behavior of three-dimensional structures relative to a certain plane, assuming that the out-of-plane effects are not important for the results. Figure 2.8 exemplifies the conception of a possible model of a steel hangar, for the analysis of the plane behavior of one of the portal frames.

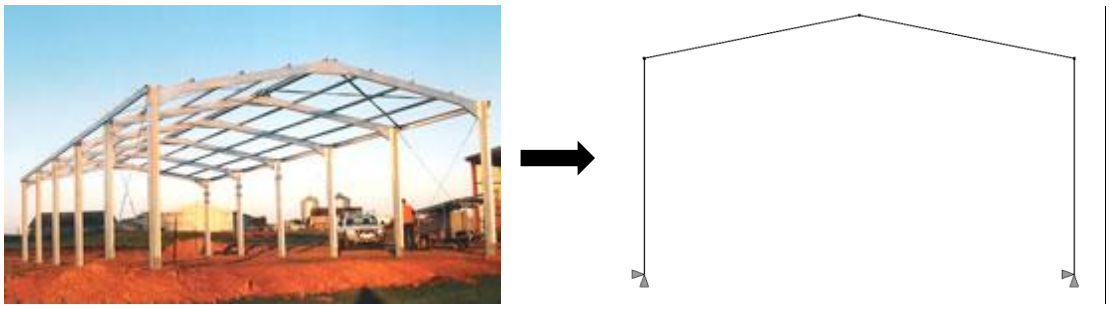

Figure 2.8 Structural model adopted to idealize the behavior of a portal frame

2.5 Grillage

A Grillage model can be seen as a plane frame model with only out-of-plane behavior. Its key features are:

- It is a two dimensional model with its elements and moment components in the XY plane, while the load components are in the global Z direction.
- By assumption, there is only out-of-plane behavior, which includes translational displacements transversal to the plane of the model, and rotations about the X and Y axes.
- Elements are laid out in a grid pattern in a single plane, rigidly connected at nodes. However, a grillage element might have a hinge (rotation liberation) at an end or hinges at both ends. In LESM, it is assumed that a hinged end releases the continuity of both bending and torsion rotations.

- In LESM, this model accepts concentrated nodal loads and moments, and distributed loads on elements. All load types are specified in the global system.
- In LESM, it is assumed that temperature gradients on elements are relative only to the local axis Z, and the axial effects of temperature variation are disregarded.
- Internal forces at any cross-section of a grillage element are: shear force, bending moment, and torsion moment. By assumption, there is no axial force in a grillage element. The convention for the positive orientation of internal forces in grillage models is represented in Figure 2.9.

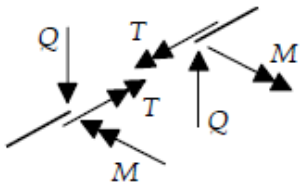

Figure 2.9 Positive orientations of internal forces in grillage models

- The geometric parameters needed for the analysis of a grillage model are the effective shear area, the bending inertia, and the torsion inertia of the cross-sections.
- Each node of a grillage model has three degrees of freedom: translation in the global Z direction, and rotation components about the global X and Y axis. The degrees of freedom numbering order considered in LESM for a grillage element is shown in Figure 2.10 in the positive directions of the local system.

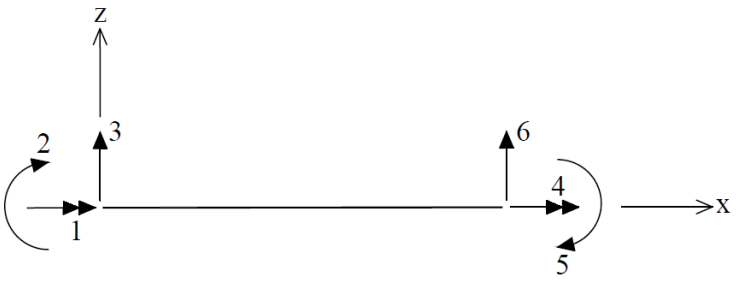

Figure 2.10 Degrees of freedom numbering order of a grillage element

- The rotation transformation matrix of a grillage element is the same of a plane frame element, as previously shown in equation 2.4.
- The considered types of support conditions are the total restriction of vertical translation, and rotations on both the global X and Y. In LESM, a rotation in a single direction cannot be released.
- This model accepts both translational and rotational prescribed displacements in the directions of fixed degrees of freedom.

A grillage model is a common type of analysis model for building stories and bridge decks. Figure 2.11 exemplifies the conception of a grillage model to analyze the transversal behavior of beams.

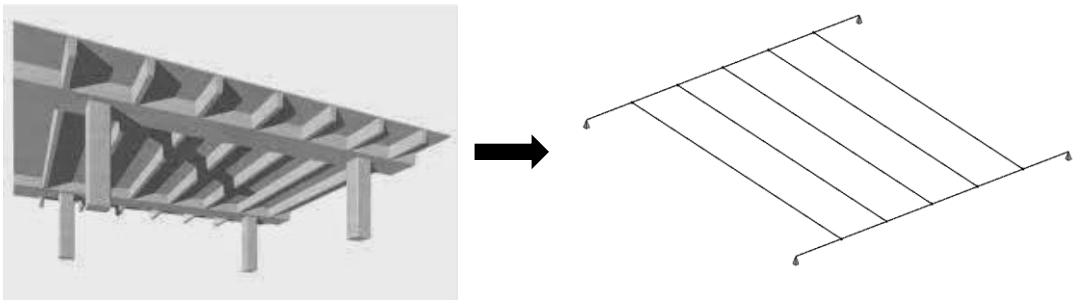

Figure 2.11 Structural model adopted to idealize the transversal behavior of beams

3 Mathematical Formulations for Linear Elements

3.1 Overview

The previous chapter presented the basic concepts of some types of linear element models and the considerations made to implement these analysis models in LESM. This chapter is dedicated to the explanation of the two theories considered in the program that are used to describe the flexural behavior of linear elements: Euler-Bernoulli (or Navier) beam theory and Timoshenko beam theory. It also provides the mathematical expressions for describing the general behavior of linear elements subjected to axial, torsional and flexural effects. These expressions, which are based on the solid mechanics theories, are necessary for the implementation of the direct stiffness method and are given in a matrix form, the same way as they are stored in the code of the program. The deduction of the formulations presented in this chapter can be found in the book *Análise Matricial de Estruturas: Aplicada a Modelos Lineares – Luiz Fernando Martha, 2016*.

3.2 Beam Theories

When considering a first-order analysis, assuming the hypothesis of small displacements, the responses of linear elements to the axial, torsional and flexural effects are independent, and the general behavior of the element is the combination of these responses. This means that the element behavior relative to each of these effects can be expressed by a different formulation. The mathematical idealization of linear elements subjected only to flexural effects (called beam elements) can be based on two distinct theories: Euler-Bernoulli beam theory (also called Navier beam theory) and Timoshenko beam theory.

In Euler-Bernoulli beam theory, it is assumed that there is no shear deformation when an element bends. As a consequence, bending of a linear element is such that its cross-section remains plane and normal to the longitudinal axis of the element. In Timoshenko beam theory, shear deformation is considered in an approximated manner, so the bending of a linear element is such that its cross-section remains plane but it is not normal to the longitudinal axis of the element.

The influence of shear deformation is smaller the greater the length of the element. Because of this, the analysis results from the two beam theories are approximately the same for elements with its length much greater than its cross-section dimensions, which is the case for most structural components. To show this, a dimensionless factor used in the formulation of Timoshenko beam theory is given in equation 3.1. This factor can be interpreted as a measure of the shear deformation relevance on the bending behavior of a beam element. In that equation, E and G are the material elasticity and shear modulus, I is the cross-section moment of inertia around the bending axis, A_s is the cross-section effective shear area obtained from the product of its full area with a shear shape factor, and L is the length of the element. This factor is inversely proportional to the square of the element length. Therefore, for slender elements its value is approximately zero, which is a consideration of Euler-Bernoulli beam theory.

$$\Omega = \frac{EI}{G A_s L^2} \tag{3.1}$$

Assuming a rectangular cross-section with a shear shape factor of 5/6 and a Poisson ratio of 0.3, the previous equation turns into equation 3.2, and the variation of

the Ω factor for different relations between the length of the element and its cross-section height, *h*, is shown in Figure 3.1.

$$\Omega = 0.26\left(\frac{L}{h}\right)^{-2} \tag{3.2}$$

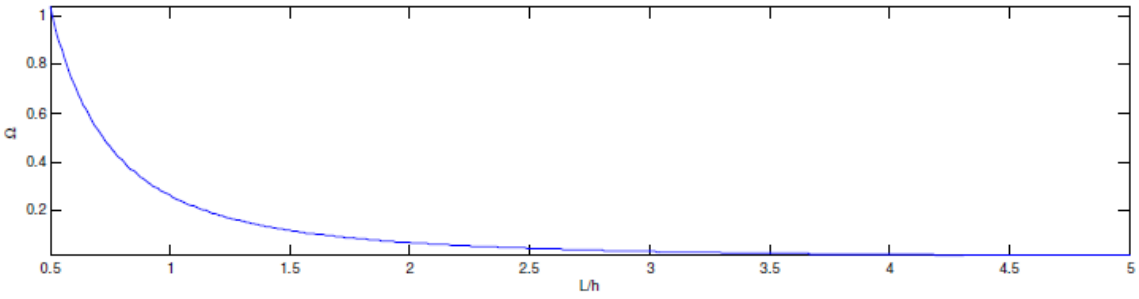

Figure 3.1 Influence of shear deformation according to the length of the elements

It can be seen that the effect of the shear deformation on the bending behavior of a beam element starts to be significant when the length is less than about twice the cross-section height, which means that the analysis results from the two theories start to diverge at this point. For a low *L/h* ratio, Timoshenko beam theory is more precise. Due to the fact that the computational effort for calculating the results by either theory is practically the same, despite the increased complexity of Timoshenko's theory, many structural analysis programs use the Timoshenko beam as the default element type.

In LESM, both beam theories can be selected to be used in the analysis process of plane frame and grillage models, whose elements have bending behavior. In truss models, the two theories may be used indistinguishably, since there is no bending behavior of an element and Euler-Bernoulli beams and Timoshenko beams are equivalent for the axial behavior.

3.3 Stiffness Matrices

The system of equilibrium equations of an element can be expressed in the matrix form of equation 3.3, where {f} is the vector of generalized forces, {d} is the vector of nodal displacements, and [k] is the element stiffness matrix.

$$\{f\} = [k]\{d\} \tag{3.3}$$

In the direct stiffness method this system is first formulated in the element local system, and then converted to the global system. The conversion of the vector of generalized forces and the vector of nodal displacements between the two reference systems was previously shown in equations 2.1 and 2.2. The conversion of the element stiffness matrix from the local to the global system is done by a triple matrix product as shown in equation 3.4, where [kel] is the element stiffness matrix in the local system and [keg] is the element stiffness matrix in the global system.

$$[keg] = [R]^T[kel][R] \tag{3.4}$$

The stiffness matrices of linear elements are symmetrical square matrices with the dimension equivalent to the number of degrees of freedom of the element: four for plane truss elements and six for plane frame and grillage elements. These matrices are assembled with the elements stiffness coefficients:

k(i,j) → Element stiffness coefficient: Forces that must act in the direction of the degree of freedom *i* to maintain the equilibrium of an isolated element when a single and unitary nodal displacement, d_j, is imposed in the direction of the degree of freedom *j*.

The value of these coefficients in the local system is presented in appendix A (Local Stiffness Coefficients of Linear Elements) in the form of decoupled matrices for each effect: axial stiffness coefficients, [kea], torsional stiffness coefficients, [ket], and flexural stiffness coefficients, [kef]. In flexural behavior, the bending can occur in the XY plane (plane frame models) or XZ plane (grillage models), and the flexural stiffness coefficients must follow the positive convention of the degrees of freedom of the corresponding model type, established in Figures 2.7 and 2.10.

The stiffness matrices of the elements of plane truss, plane frame and grillage models are presented in equations 3.5 to 3.7, respectively, in the local system. Because of the independent behavior to the effects in different directions, these matrices can be assembled using the decoupled stiffness coefficients matrices, resulting in null coefficients in some positions of the element stiffness matrix.

$$[\mathrm{kel}] = \begin{bmatrix} \mathrm{kea}(1,1) & 0 & \mathrm{kea}(1,2) & 0 \\ 0 & 0 & 0 & 0 \\ \mathrm{kea}(2,1) & 0 & \mathrm{kea}(2,2) & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.5}$$

$$[\mathrm{kel}] = \begin{bmatrix} \mathrm{kea}(1,1) & 0 & 0 & \mathrm{kea}(1,2) & 0 & 0 \\ 0 & \mathrm{kef}(1,1) & \mathrm{kef}(1,2) & 0 & \mathrm{kef}(1,3) & \mathrm{kef}(1,4) \\ 0 & \mathrm{kef}(2,1) & \mathrm{kef}(2,2) & 0 & \mathrm{kef}(2,3) & \mathrm{kef}(2,4) \\ \mathrm{kea}(2,1) & 0 & 0 & \mathrm{kea}(2,2) & 0 & 0 \\ 0 & \mathrm{kef}(3,1) & \mathrm{kef}(3,2) & 0 & \mathrm{kef}(3,3) & \mathrm{kef}(3,4) \\ 0 & \mathrm{kef}(4,1) & \mathrm{kef}(4,2) & 0 & \mathrm{kef}(4,3) & \mathrm{kef}(4,4) \end{bmatrix} \tag{3.6}$$

$$[\mathrm{kel}] = \begin{bmatrix} \mathrm{ket}(1,1) & 0 & 0 & \mathrm{ket}(1,2) & 0 & 0 \\ 0 & \mathrm{kef}(2,2) & \mathrm{kef}(2,1) & 0 & \mathrm{kef}(2,4) & \mathrm{kef}(2,3) \\ 0 & \mathrm{kef}(2,2) & \mathrm{kef}(1,1) & 0 & \mathrm{kef}(1,4) & \mathrm{kef}(1,3) \\ \mathrm{ket}(2,1) & 0 & 0 & \mathrm{ket}(2,2) & 0 & 0 \\ 0 & \mathrm{kef}(4,1) & \mathrm{kef}(4,1) & 0 & \mathrm{kef}(4,4) & \mathrm{kef}(4,3) \\ 0 & \mathrm{kef}(3,2) & \mathrm{kef}(3,1) & 0 & \mathrm{kef}(2,4) & \mathrm{kef}(3,3) \end{bmatrix} \tag{3.7}$$

3.4 Fixed End Forces

Fixed end forces are the support reactions of isolated elements subjected to external solicitations. In the direct stiffness method these reactions correspond to the equivalent nodal loads, coming from internal loads or temperature variation of elements, in the opposite direction.

In LESM, the formulations for fixed end forces consider a linearly distributed load with axial and transversal components, which is the most general case for a distributed load solicitation. It also considers a uniform temperature variation along the superior and inferior faces of the elements, which can lead to an axial temperature variation and a temperature gradient.

The value of the fixed end forces for each element continuity condition, in the local system, is presented in appendix B (Fixed End Forces for Linearly Distributed Load) and C (Fixed End Forces for Temperature Variation). These values, just like the stiffness coefficients values, are given in the form of decoupled vectors for each individual effect: axial fixed end forces, [fea], and flexural fixed end forces, [fef]. There is no torsional reaction because distributed torsion moments are not supported by the program. The flexural behavior distinguishes the bending in the XY plane (plane frame models) of the XZ plane (grillage models). The reactions obey the positive convention of the degrees of freedom of the corresponding model type, established in Figures 2.7 and 2.10.

The vectors with fixed end forces values are shown in equations 3.8 to 3.10 for elements of plane truss, plane frame and grillage models respectively. These vectors are assembled using the decoupled fixed end forces vectors for each effect, where:

fel(i) → Element local fixed end force: Support reaction of an element, subjected to an internal solicitation, in the direction of the degree of freedom *i* in the local system.

$$[\mathrm{fel}] = \begin{bmatrix} \mathrm{fea}(1) \\ \mathrm{fef}(1) \\ \mathrm{fea}(2) \\ \mathrm{fef}(3) \end{bmatrix} \tag{3.8}$$

$$[\mathrm{fel}] = \begin{bmatrix} \mathrm{fea}(1) \\ \mathrm{fef}(1) \\ \mathrm{fef}(2) \\ \mathrm{fea}(2) \\ \mathrm{fef}(3) \\ \mathrm{fef}(4) \end{bmatrix} \tag{3.9}$$

$$[\mathrm{fel}] = \begin{bmatrix} \mathrm{fea}(1) \\ \mathrm{fef}(2) \\ \mathrm{fef}(1) \\ \mathrm{fea}(2) \\ \mathrm{fef}(4) \\ \mathrm{fef}(3) \end{bmatrix} \tag{3.10}$$

Eventually these vectors must be converted to the global system. This is done by using the element rotation transformation matrix as shown in equation 3.11, where [fel] is the fixed end forces vector in the local system and [feg] is the fixed end forces vector in the global system.

$$[\mathrm{feg}] = [R]^T [\mathrm{fel}] \tag{3.11}$$

3.5 Shape Functions

Shape functions describe the deformed configuration of an element subjected to a single and unitary nodal displacement by giving the axial and transversal displacements along its longitudinal axis. These functions are not used in the direct stiffness method, but they are necessary to graphically represent the deformed configuration of plane frame and grillage models. Plane truss models do not make use of shape functions because in this case the elements always remain straight, so the deformed configuration can be obtained only by the coordinates of the displaced nodes.

$N_i(x)$ → Element shape function: Function that describes the deformed configuration of an isolated element when a single and unitary nodal displacement d_i is imposed in the direction of the degree of freedom *i*.

With shape functions, the deformed configuration of an isolated element can be described by an interpolation of its initial and final nodes displacements. Equations 3.12 and 3.13 give the axial, *u(x)*, and transversal, v*(x)*, displacements of an element in terms of its shape functions and nodal displacements.

$$u(x) = N_1(x).d_1 + N_4(x).d_4 \tag{3.12}$$

$$v(x) = N_2(x).d_2 + N_3(x).d_3 + N_4(x).d_4 + N_5(x).d_5 \tag{3.13}$$

These equations can be expressed in a matrix form, as equations 3.14 and 3.15, where [N] is the shape functions matrix.

$$\begin{bmatrix} u(x) \\ v(x) \end{bmatrix} = \begin{bmatrix} N_1(x) & 0 & 0 & N_4(x) & 0 & 0 \\ 0 & N_2(x) & N_3(x) & 0 & N_5(x) & N_6(x) \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} \tag{3.14}$$

$$\{u\} = [N]\{d\} \tag{3.15}$$

The expressions for the shape functions are presented in appendix D (Shape Functions) for each element end continuity condition.

3.6 Internal Displacements in Loaded Elements

In plane frame and grillage models, the deformed configuration of an element considers not only the displacements of the end nodes, but also the internal displacements caused by distributed loads and temperature variation. The expressions that describe the axial and transversal displacement components from these two effects are given in Appendix E (Internal Displacement from Linearly Distributed Load) and Appendix F (Internal Displacement from Temperature Variation). To determine the total displacement at any cross-section of an element, these expressions are then added to the ones that provide the internal displacements in terms of shape functions and nodal displacements.

4 Computational Implementation

4.1 Overview

Structural analysis methods can be divided into two groups: classic methods and matrix methods. The second is more appropriate for computational implementation because the emphasis is in the generalization and, as operating with matrices in high level computer languages is very simple, it allows the automation of the process. The purpose of the previous chapter in assembling all coefficients and equations in matrices and vectors is to make them usable by the direct stiffness method. This matrix-based method is implemented in LESM and many commercial programs for calculating element forces and displacements in structures using stiffness relations.

Developing a structural analysis program to assist the work of professionals requires more than just implementing the code for calculating analysis results. This chapter focuses on explaining some concepts of computer science that were used in the development of all the processing stages of LESM, and could be used for developing any other structural analysis program.

4.2 Implementation stages of a typical graphic program of structural analysis

The step of calculating the analysis results, given all the required information about the structural model, is called data processing. This procedure is nothing more than a series of instructions (algorithm) written in a particular programming language (code) to make it possible the interpretation of the input data and the calculation of the results, through an input/output (I/O) operation. When developing a computer program for users, it becomes an application, and a few more procedures are required

to make the program usable and provide an interactive experience to users: the data pre-processing, the data post-processing, and the development of a Graphical User Interface (GUI).

The pre-processing stage consists in providing input data with the model information and preparing it to be used in the data processing stage. This data provision can be done through the process of modeling or loading an input file. Modelling allows users to build a structural model by gradually providing information about materials, cross-sections, nodes, elements, support conditions, etc. An input file contains the full description of a model in a format that the program can read and interpret.

The post-processing stage is responsible for making the output data from the analysis process available to users by printing or graphically displaying them on the screen of the computer. In the case of a structural analysis program, the visible results are internal forces diagrams and deformed configurations. This stage is crucial to make the data output easily interpretable to users, letting them see these results and control the visualization.

In order to implement the pre-processing and the post-processing stages of a program, a GUI must be developed to allow users to interact with the code through mouse control, keyboard entry, and graphical components such as push-buttons, checkboxes, sliders, etc. The development of programs with an interactive graphical interface requires two basic tools: a Graphic System and a User Interface System. The first has the function to generate images in windows managed by the second. That is, the interface system is responsible for the architecture and behavior of the GUI by managing components like windows, buttons and menus, while the graphic system is

responsible for drawing geometric primitives like points, lines and shapes to represent the structural model in an interface component called canvas.

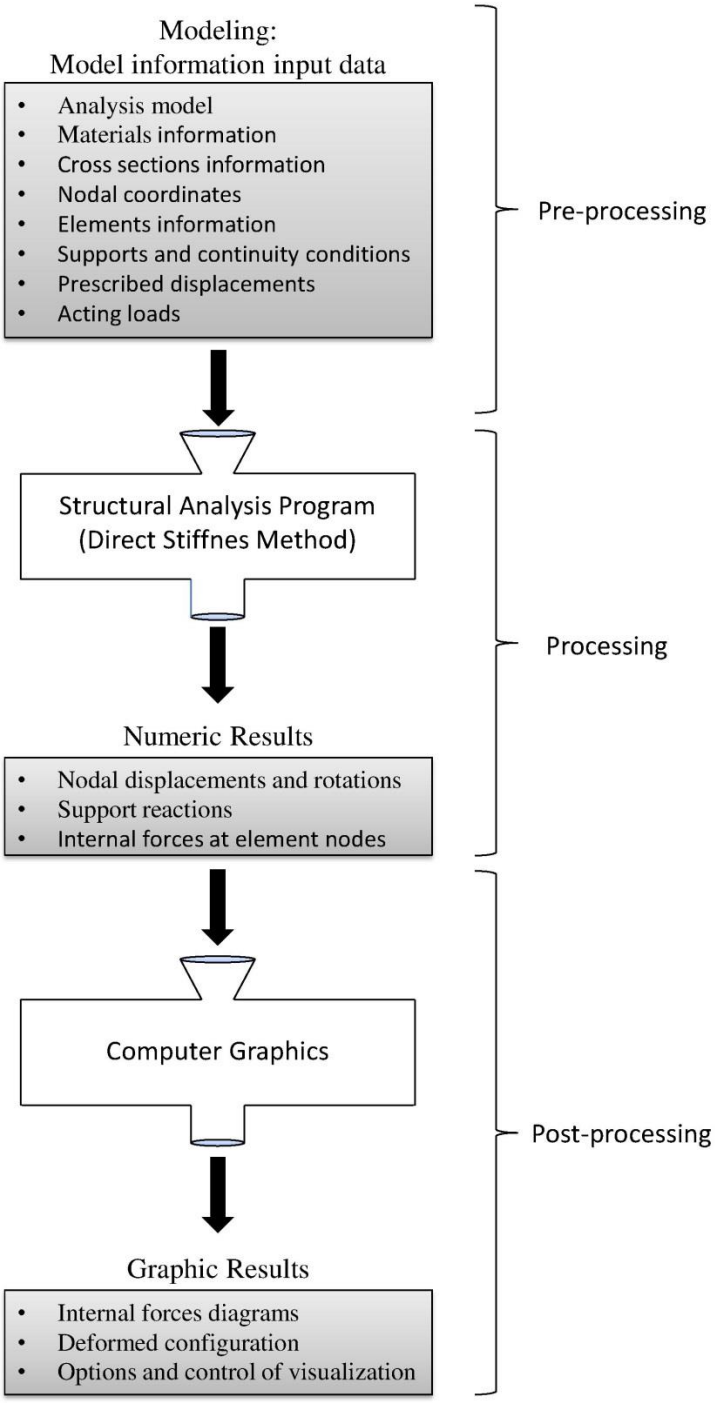

Figure 4.1 Flowchart of the operating stages of LESM

4.3 The Use of MATLAB to Develop a Graphic Program

MATLAB (short for Matrix Laboratory) is a powerful software package consisting in an interactive development environment that has built-in functions and tools to accomplish a diverse range of tasks such as mathematical operations, programming, and graphical illustrations that can be used to develop applications efficiently. This section will present the main characteristics of MATLAB and the reasons why it was chosen to develop LESM.

The great differential of MATLAB is that it works only with matrices, so even a scalar value is stored in a 1x1 matrix variable. Because of this, there are many operations and functions that can be performed on entire vectors and matrices without running a loop, as if they were simple numeric variables. Also, the type of the variables does not need to be specified. These aspects simplify the code, make it much cleaner and understandable, and it is a great advantage when writing a program that makes use of several matrix operations, such as the algorithm to implement the direct stiffness method, especially when the focus of the code is to have a didactic approach.

Besides the facilities with mathematical operations, MATLAB is also a high level programming language. This means that it aims to provide a higher level of abstraction of the internal computer hardware details, making its commands to be very similar to human writing. The computer, however, can only interpret commands written in machine language, so the code must be translated before the computer can actually execute the sequence of instructions in the program. In MATLAB this translation is done by an *interpreter* that goes through the code, line-by-line, translating and executing each command.

MATLAB is a multi-paradigm programming language, that is, a programming language that supports more than one programming paradigm, such as the *object-oriented* and the *event-driven*, both used in different development stages of LESM and described in the following sections.

In addition to a clear code, it is very important that all files are well documented, so people can easily understand what each step does when analyzing the code. A common way to document the code is to include comments and, in the case of MATLAB, it is also possible to publish the files. The *publish* command creates a readable, organized and formatted document that includes codes, comments, and data output. It is a very useful tool for sharing the code as it can be converted to different document formats. The link to the *html* publication of the processing stage of LESM is given in section 1.2 (Objectives).

The implementation of the post-processing stage and the generation of graphic results are also of a great simplicity when using MATLAB. Its graphic system has many graphing capabilities, including 2D and 3D plotting functions whose use is facilitated by the possibility of passing vectors and matrices as parameters. Functions to control the camera and the model visualization do not need to be implemented, since MATLAB already include toolbars that allow this task.

Finally, there are two ways to develop a GUI in MATLAB: programmatically or using the Graphical User Interface Development Environment (GUIDE). The latter was chosen to develop an interface for LESM and it is based on the *event-driven* programming paradigm. The *object-oriented* programming paradigm is used in the code of the processing stage and implicitly used in the post-processing stage because graphic objects are created when a plotting function is called.

4.4 Object-Oriented Programming Paradigm

Differently from the procedural programming paradigm, which consists in a list of instructions with no association between functions and the operated data, the object-oriented programming (OOP) paradigm is based on classes equipped with data fields (properties) and associated functions (methods). The most important distinction is that procedural programming uses procedures to operate on data structures, whereas the OOP paradigm bundles the two together, increasing code understanding and facilitating its maintenance and expansion. This would allow the extension of LESM data processing to support spatial analysis models, for example, with greater simplicity.

In the OOP paradigm, a class is a template for defining properties and methods, as well as default values and behavior. Instances of a class, also called objects, can be created by setting the properties values. Each object has its behavior controlled by the methods defined by its class.

A class definition consists in two main blocks, one with the definition and initialization of the properties, and the other with the implementation or declaration of the methods. There are two methods that should always be implemented in a class definition, the constructor method and the destructor method. The first must have the same name of the class and it is required to create objects of that class with prescribed values for their properties. The destructor method is used to clean the objects by resenting their properties values.

The basic characteristics of the OOP paradigm are:

- Inheritance: This mechanism allows one class to be derived from another, so that the initial class is called superclass and the derived class is called subclass. The subclass is a new class that implements the methods declared in its superclass. This technique is indicated when the same procedures are being used several times.
- Abstraction: An abstract class cannot be instantiated and it is usually a superclass that declares the methods that must be implemented in its subclasses. These methods declared in the superclass and implemented in the subclasses are called abstract methods. When the implementation of a method is independent of the subclass, it can be done in the superclass.
- Polymorphism: This concept, used by abstract classes, allows methods of two or more subclasses, derived from the same superclass, to have the same name but different implementation. With this mechanism, the same method can be implemented in different subclasses and the superclass knows which one to call depending on the subclass of the object.
- Encapsulation: This concept is related to data protection, keeping it safe from outside interference and misuse. This occurs when a class does not allow the code to access the internal data of its objects or when this access is given only through specific methods. This mechanism of the OOP paradigm makes it a better and safer alternative than procedural programming, but it assumes that users should only need to know the functionality of methods, not how it is implemented. In LESM, however, the goal is to preserve the simplicity of the code, so the access to the methods and properties of the objects is not restricted.

Attributes are a group of characteristics inherent to a class, method or property that can be set to modify their behavior. When an attribute need to be different from the default setting, it must be specified in the beginning of the class definition, or in the beginning of the properties or methods blocks. To give different attributes to properties or methods it is necessary to split them into blocks with the same attribute setting. Attributes of classes, properties and methods are not inherited, so the settings of a superclass do not affect its subclasses.

The access to properties and methods of a class can be controlled by setting their accessibility attribute, which is used to apply the concept of encapsulation. This attribute can be set to three options: public (access is possible from anywhere), private (access is possible only in the respective class), protected (access is possible only in the respective class or subclass). In LESM the accessibility attribute of all properties and methods is set to public. Another method attribute that is commonly set is the definition of an abstract method or a static method. The first is used to declare methods of an abstract class and the second specifies methods that do not depend on any object.

Allowing the free manipulation of data in LESM makes methods able to call properties and other methods by using the dot operator and the name of the object as prefix. To access a property or a method of its own object the keyword *obj* is used as prefix. Another way to make these calls is using the *set* and *get* methods that are implicitly called when a reference to a property or method is made using the dot operator. The *set* and *get* methods are implemented in LESM by deriving the created classes from built-in handle classes.

There are two types of classes in MATLAB: value classes and handles classes. The first is the default type and its objects are associated with the variable to which it is assigned. If this variable is reassigned, an independent copy of the original object is created, so that any change to this copy does not affect the original object. If a variable containing an object of a value class is passed to a function for the purpose of modifying it, this function must return the modified object as an output argument.

Handle classes, on the other hand, originate handle objects. When these objects are assigned to variables or passed to a function, their data is not copied; instead, a reference is created. Therefore, it is possible to assign a handle object to multiple variables, pass it to functions without making a copy of the original object, and it is not needed that functions return modified objects. Handle classes allow more complex interactions among objects because they can reference each other, and they are also used to create graphic objects and figure windows in MATLAB, since each visual element is a handle object. All handle classes must be derived from a built-in abstract handle class of MATLAB.

In LESM, all classes are handle classes derived from the MATLAB built-in abstract class *matlab.mixinSetGet*. This built-in class derives handle classes that inherit the *set* and *get* methods, responsible for assigning or returning values from a specific property. These methods are called when a property is referenced using the dot operator and they must be an implemented for each property of a class.

4.5 Event-Driven Programming Paradigm

The event-driven programming paradigm works with the idea that actions are triggered by events, so the program is designed to react. These events can be mouse clicks, key presses, a selection on a drop-down list, an entry into a text box, etc. They determine the flow of the program by calling specific functions, called *callback functions*, with the code implementation responsible for executing the action related to a specific event.

This programming paradigm is largely used in the development of graphical interfaces, since the code that manages the interface behavior is supposed to perform certain procedures in response to user actions. The interface of LESM was created using the Graphical User Interface Development Environment (GUIDE), a tool that is a drag-and-drop environment for laying out user interfaces that can display any type of MATLAB plot and contains various interactive components, including push-buttons, menus, tool bars, tables, etc. MATLAB automatically generates a structured file with the *callback functions* related to the actions that can be executed in each of the components added to the interface layout.

5 The Development of LESM

5.1 Overview

The previous chapters presented some concepts of structure analysis and computer science required to understand the implementation of a structural analysis program, including comments of how these concepts are considered and implemented in LESM. This chapter combines these two areas and describes the development of the operating stages and the interface of the program. It also gives a complete description of all class created with the OOP paradigm.

5.2 The Classes of LESM

The program is divided in nine classes, seven of them are related to the processing stage, to create a structural model and calculate the analysis results. The other two are associated with the post-processing stage, to draw or print these results on the screen of the computer.

As mentioned in chapter 4 (Computational Implementation), the accessibility attribute of all properties and methods in LESM are set to public, so that objects can easily interact with each other and the code becomes cleaner and simpler to understand. This enables the properties of all objects to be accessed and modified by methods of any object or functions of any file. The call of a property or method can be done with the dot operator, which implicitly calls the *set* or *get* methods, inherited from a built-in class in MATLAB responsible for deriving handle classes. By making the classes of the program as handle classes, it is possible to reference objects instead of creating copies to be passed between methods and functions.

The flowchart in Figure 5.1 illustrates the interaction between all implemented classes. Each box represents an object of a class without distinguishing if it is a superclass or a subclass. The arrows indicate the handle properties of these objects i.e., if an object of class “A” points to an object of class “B”, it means that the first has a property that is a handle to the second. For example, if an object of the *Draw* class needs to access the initial and final nodal coordinates of a specific element, it has to make a reference chaining with objects of the *Drv*, *Elem* and *Node* classes, to access the property of the node object that gives its coordinates.

The blue color is associated with objects related to the processing stage, while the red color is for objects used in the post-processing stage.

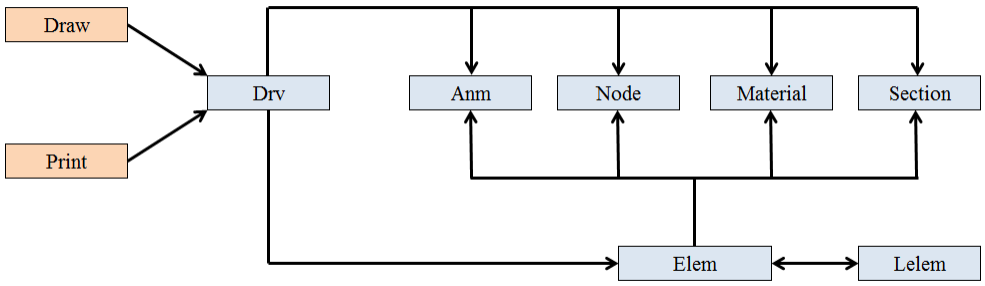

Figure 5.1 Flowchart of the classes of LESM

In the following sections it will be described the characteristics of each class, together with a table containing the name and a short explanation of all its properties and methods. The *set*, *get*, *constructor* and *clean* methods are not included because they have the same functionality in all classes and were previously explained.

5.2.1 The *Drv* (Driver) Class

This class is the one that drives the analysis process. Its properties store the general information about the structural model and its methods are functions that correspond to the main steps of the direct stiffness method. Since all data of a model can be accessed using the properties of an object of the *Drv* class, only one object of this class is created during the processing stage.

Drv Class	
Properties	Description
anm	Handle to an object of the Anm class
materials	Vector of handles to objects of the Material class
sections	Vector of handles to objects of the Section class
nodes	Vector of handles to objects of the Node class
elems	Vector of handles to objects of the Elem class
K	Global stiffness matrix
F	Global forces vector
D	Global displacements vector
ID	Degree of freedom numbering matrix
nmat	Number of materials
nsec	Number os cross-sections
nnp	Number of nodal points
nel	Number of elements
neq	Number of equations
neqfree	Number of equations of free degrees of freedom
neqfixed	Number of equations of fixed degrees of freedom
Static Methods	Description
openFile	Opens input file
Public Methods	Description
fictRotConstraint	Inserts or removes a fictitious rotation constraint on nodes
dimKFD	Dimensions the global stiffness matrix, forces vector and displacements vector
assembleDOFNum	Assembles the degree of freedom numbering matrix
assembleGle	Assembles the gather vector of an element (stores element d.o.f. numbers)
gblMtx	Assembles the global stiffness matrix
assembleElemMtx	Assembles the stiffness matrix of an element to the global stiffness matrix
elemLoads	Adds the equivalent nodal loads of an element to the global forces vector
assembleENL	Assembles the equivalent nodal load vector of an element to the global forces vector
solveEqnSystem	Partitions and solves the system of equations
elemIntForce	Computes internal forces of each element
process	Processes current model data according to the direct stiffness method

Figure 5.2 Properties and methods of the *Drv* class

5.2.2 The *Anm* (Analysis Model) Class

The *Anm* class is an abstract superclass that declares abstract methods in which the implementation depends on the analysis model. The implementation of these abstract methods is done in the subclasses, each one considering the specific behavior of the corresponding analysis model. The behavior of the three analysis models supported by LESM was explained in chapter 2 (Linear Element Models). The subclasses derived from this superclass are: *Anm_Truss2D*, *Anm_frame2D* and *Anm_Grillage*.

As the value of the properties is always the same for objects of the same *Anm* subclass, the use of this class is based on its methods. For this reason, only one object is created in the *processing* stage, which is used to call its methods.

Anm Class	
Properties	Description
analysis_type	Type of analysis model
ndof	Number of degrees of freedom per node
Abstract Methods	Description
gblToLocElemRotMtx	Assembles the rotation transformation matrix
setupDOFNum	Sets up the degree of freedom numbering matrix
setupPrescDispl	Stores prescribed displacements in the global displacements vector
elemLocStiffMtx	Assembles the stiffness matrix of an element
nodalLoads	Adds nodal load components to the global forces vector
elemLocUniformLoadFEF	Assembles the fixed end force vector of an element for a uniformely distributed load
elemLocLinearLoadFEF	Assembles the fixed end force vector of an element for a linearly distributed load
elemLocThermalLoadFEF	Assembles the fixed end force vector of an element for thermal expansion
initIntForce	Initializes the internal forces vectors of an element with null values
assembleIntForce	Assembles the internal forces vectors of an element

Figure 5.3 Properties and methods of the *Anm* class

5.2.3 The *Elem* (Element) Class

This is an abstract superclass that generically defines a linear element object. It specifies the properties of an element, implements methods related to the generic behavior of linear elements, and declares methods that depend on the element type.

The properties of this superclass consider a three-dimensional behavior of an element in its local axes system. An object of the *Anm* class is responsible to "project" this general three-dimensional behavior to a specific model behavior. Therefore, one of the properties of an object of the *Elem* class is a handle to a target *Anm* object.

As explained in section 3.2 (Beam Theories), linear elements can be divided into two types according to the theory considered for the flexural behavior. The generic behavior of linear elements considers everything that does not depend on bending, which are the axial and torsional effects. Therefore, derived subclasses should implement abstract methods that deal with flexural behavior, whereas methods related to general behavior of elements are implemented in the superclass itself. The subclasses derived from this superclass are:

- *Elem_Navier*: This subclass deals with Navier (Euler-Bernoulli) flexural behavior of linear elements. In Euler-Bernoulli flexural behavior, it is assumed that there is no shear deformation. As a consequence, bending of a linear element is such that its cross-section remains plane and normal to the element longitudinal axis.
- *Elem_Timoshenko*: This subclass deals with Timoshenko flexural behavior of linear elements. In Timoshenko flexural behavior, shear deformation is considered in an approximated manner. Bending of a linear element is such that its cross-section remains plane but it is not normal to the element longitudinal axis.

Because elements are considered to be three-dimensional, each abstract method that deals with flexural behavior has two implementations, one considering the bending in the local plane XY (plane truss and plane frame models) and the other

considering the bending in the local plane XZ (grillage models). The differences between these implementations are in the distinct degrees of freedom involved in each analysis model and the cross-section properties that are used.

This superclass does not deal with properties and methods that are related to element loads. Therefore, one of the properties of an object of the *Elem* class is a handle to an object of the *Lelem* class that contains all the information about the loads acting on the element.

Elem Class	
Properties	Description
nen	Number of nodes (always 2)
anm	Handle to the object of the Anm class
type	Element type (Euller-Benoulli or Timoshenko)
material	Handle to an object of the Material class
section	Handle to an object of the Section class
nodei	Handle to an object of the Node class corresponding to the initial node
nodef	Handle to an object of the Node class corresponding to the final node
hingei	Flag for hinge at initial node
hingef	Flag for hinge at final node
length	Length value
cosine_X	Orientation cosine with global axis X
cosine_Y	Orientation cosine with global axis Y
cosine_Z	Orientation cosine with global axis Z
rot	Rotation transformation matrix
gle	Gather vector (stores element d.o.f. numbers)
load	Handle to an object of the Lelem class
axial_force	Vector of axial internal forces at element ends
shear_force_Y	Vector of shear forces at element ends relative to local axis y
shear_force_Z	Vector of shear forces at element ends relative to local axis z
bending_moment_Y	Vector of bending moments at element ends relative to local axis y
bending_moment_Z	Vector of bending moments at element ends relative to local axis z
torsion_moment	Vector of torsion moments at element ends
Abstract Methods	Description
flexuralStiffCoeff_XY	Generates the flexural stiffness coefficient matrix relative to local plane XY
flexuralStiffCoeff_XZ	Generates the flexural stiffness coefficient matrix relative to local plane XZ
shapeFunctionMtx_XY	Generates the shape functions matrix relative to local plane XY
shapeFunctionMtx_XZ	Generates the shape functions matrix relative to local plane XZ
Public Methods	Description
gblStiffMtx	Generates the stiffness matrix in the global system
axialStiffCoeff	Generates the axial stiffness coefficient matrix
torsionStiffCoeff	Generates the torsion stiffness coefficient matrix
gblAnlIntForce	Gets the internal force vector caused by end nodes displacements

Figure 5.4 Properties and methods of the *Elem* class

5.2.4 The *Lelem* (Load Element) Class

The *Lelem* class is an abstract superclass that specifies the load components of a linear element object, implements methods related to the generic behavior of linear elements, and declares methods that depend on the element type. The generic behavior of linear elements and the element types were briefly explained in the previous section. This class considers load components that act in three dimensions.

Differently from the *Elem* class, where methods are functions that depend on the physical and geometric properties of elements, the methods of this class are those that make use of load information, such as the calculation of fixed end forces (FEF), equivalent nodal loads (ENL) and internal displacements. To access the information about the element where these loads are applied, an object of the *Lelem* class must have a property that is a handle to the associated element object.

Just like in the *Elem* class, subclasses of the *Lelem* superclass are responsible for the implementation of the abstract methods that deal with flexural behavior. These subclasses are: *Lelem_Navier* and *Lelem_Timoshenko*. Each abstract method is implemented considering both the bending in the local plane XY (plane truss and plane frame models) and the bending in the local plane XZ (grillage models) because of the distinct load components and cross-section properties used in each case.

Lelem Class	
Properties	Description
elem	Handle to an object of the Elem class
unifDir	Flag for the uniform load direction (local or global system)
uniformGbl	Vector of uniformly distributed load components in global system
uniformLcl	Vector of uniformly distributed load components in local system
linDir	Flag for the linear load direction (local or global system)
linearGbl	Vector of linearly distributed load components in global system
linearLcl	Vector of linearly distributed load components in local system
tempVar_X	Temperature variation in the center of gravity
tempVar_Y	Temperature gradient relative to local axis Y
tempVar_Z	Temperature gradient relative to local axis Z
Abstract Methods	Description
flexuralLinearLoadFEF_XY	Generates the flexural FEF vector for a linearly distributed load in local plane XY
flexuralLinearLoadFEF_XZ	Generates the flexural FEF vector for a linearly distributed load in local plane XZ
flexuralThermalLoadFEF_XY	Generates the flexural FEF vector for a temperature variation in local plane XY
flexuralThermalLoadFEF_XZ	Generates the flexural FEF vector for a temperature variation in local plane XZ
locDisplLinearLoad_XY	Generates the internal displacements vector for a linearly distributed load in local plane XY
locDisplLinearLoad_XZ	Generates the internal displacements vector for a linearly distributed load in local plane XZ
locDisplTempVar_XY	Generates the internal displacements vector for a temperature variation in local plane XY
locDisplTempVar_XZ	Generates the internal displacements vector for a temperature variation in local plane XZ
Public Methods	Description
axialLinearLoadFEF	Generates the axial FEF vector for a linearly distributed load
axialThermalLoadFEF	Generates the axial FEF vector for a temperature variation
simplySuppLinearLoadFEF	Generates the transversal FEF vector for a linearly distributed load (simply support at both ends)
gblUniformLoadENL	Generates the equivalent nodal load vector for a uniformly distributed load
gblLinearLoadENL	Generates the equivalent nodal load vector for a linearly distributed load
gblThermalLoadENL	Generates the equivalent nodal load vector for a temperature variation

Figure 5.5 Properties and methods of the *Lelem* class

5.2.5 The *Node* Class

The *Node* class specifies the properties of nodal points considering a three-dimensional space. An object of this class has three coordinates and six degrees of freedom. The vectors of essential boundary conditions, prescribed displacements and nodal loads have six positions, each one associated with the degree of freedom of the same number. Even though a value is assigned to all positions of these vectors, only the values associated with the degrees of freedom of the respective analysis model are used.

Node Class	
Properties	Description
id	Identification number
coord_X	Coordinate on global axis X
coord_Y	Coordinate on global axis Y
coord_Z	Coordinate on global axis Z
ebc	Vector of essential boundary conditions flags
prescDispl	Vector of prescribed displacements values
nodalLoad	Vector of nodal loads components
Public Methods	Description
elemsIncidence	Counts total number of elements connected to a node

Figure 5.6 Properties and methods of the *Node* class

5.2.6 The *Material* Class

The *Material* class specifies the properties of a homogeneous and isotropic material. This class has no method other than those implemented in all classes.

Material Class	
Properties	Description
id	Identification number
elasticity	Elasticity modulus
poisson	Poisson ratio
shear	Shear modulus
thermexp	Thermal expansion coefficient

Figure 5.7 Properties and methods of the *Material* class

5.2.7 The *Section* Class

This class defines a three-dimensional generic cross-section by specifying all its properties, even though some of them are not used depending on the analysis model. This class also does not have any implemented method besides those required methods.

Section Class	
Properties	Description
id	Identification number
area_X	Area relative to local axis X (full area)
area_Y	Area relative to local axis Y (effective shear area)
area_Z	Area relative to local axis Z (effective shear area)
inertia_X	Moment of inertia relative to local axis X (torsion inertia)
inertia_Y	Moment of inertia relative to local axis Y(bending inertia)
inertia_Z	Moment of inertia relative to local axis Z (bending inertia)
cw	Warping torsional constant
height_Y	Height relative to local axis Y
height_Z	Height relative to local axis Z

Figure 5.8 Properties and methods of the *Section* class

5.2.8 The *Draw* Class

This class is an abstract superclass that defines an object for drawing graphic entities, implements static methods for plotting geometric figures, and declares abstract methods whose implementation depends on the analysis model. The derived classes that implement these abstract methods are: *Draw_Truss2D, Draw_Frame2D* and *Draw_Grillage*.

Only one object of this class is created for the post-processing stage and its methods are used to display the structural model and the analysis results on the screen, as well as set some visualization parameters. For this reason, this object needs to have a property that is a handle to an object of the *Drv* class that contains all the information about the model and the results.

The static methods were implemented to generalize the code for plotting geometric figures that are commonly used for drawing the model and the results. Calling these methods, instead of writing the code for plotting the same shapes multiple times, simplifies the implementation of the abstract methods.

Draw Class	
Properties	Description
drv	Handle to an object of the Drv class
size	Drawing size parameter
az	Canvas viewpoint azimuth
elev	Canvas viewpoint elevation
Static Methods	Description
circle	Plots a circle with defined position, size and color
sphere	Plots a sphere with defined position and size
square	Plots a square with defined position, size and color
cube	Plots a cube with defined position, size and color
triangle	Plots a triangle with defined position, size, orientation and color
pyramid	Plots a pyramid with defined position, size, orientation and color
arrow2D	Plots a plane arrow with defined position, size, orientation and color
arrow3D	Plots a spatial arrow with defined position, size, orientation and color
moment2D	Plots a plane moment symbol with defined position, size, orientation and color
moment3D	Plots a spatial moment symbol with defined position, size, orientation and color
Abstract Methods	Description
setSize	Sets viewpoint position and drawing size parameter
setLimits	Sets axes limits
scaleFactor	Sets scale factor value
model	Draws structural model (only with nodes, supports and elements)
elements	Draws elements with hinged or continuous ends
nodes	Draws nodal points with support conditions
elemLoads	Draws distributed loads (uniform and linear)
nodalLoads	Draws applied nodal loads and moments
nodalPrescDispl	Draws indication of nodal prescribed displacements
thermalLoads	Draws thermic variation representation
nodeID	Plots identification number of nodes
elementID	Plots identification number of elements
elementOrientation	Draws indication of elements orientation
deformConfig	Draws the structure deformed configuration
axialForce	Draws the resulting axial force diagram
shearForce	Draws the resulting shear force diagram
bendingMoment	Draws the resulting bending moment diagram
torsionMoment	Draws the resulting torsion moment diagram
reactions	Draws indication of support reactions

Figure 5.9 Properties and methods of the *Draw* class

5.2.9 The *Print* Class

The *Print* class is an abstract superclass similar to the *Draw* class in terms of use and applicability that instead of defining an object for drawing graphic entities, it defines an object for printing the information about the model and the results in a textual output file during the post-processing stage. This class implements public methods

and declares abstract methods for printing text information that depends on the analysis model. The subclasses derived from this superclass are: *Print_Truss2D*, *Print_Frame2D* and *Print_Grillage*.

A single object of this class is created to allow the use of its methods. One of the properties of this object is a handle to an object of the *Drv* class that contains all the information about the model and the results.

Print Class	
Properties	Description
drv	Handle to an object of the Drv class
txt	Identifier of the output file
Abstract Methods	Description
results	Prints analyis results
analysisLabel	Prints the analysis model type
nodalSupport	Prints information about support conditions
nodalPrescDisp	Prints information about prescribed displacements
nodalLoads	Prints information about nodal loads
elements	Prints information about elements
unifElementLoads	Prints information about uniform loads
linearElementLoads	Prints information about linear loads
temperatureVariation	Prints information about temperature variation
nodalDisplRot	Prints results of nodal displacement/rotation
reactions	Prints results of support reactions
intForces	Prints results of internal forces at element nodes
Public Methods	Description
header	Prints analysis results header
modelDescrip	Prints global description of model
material	Prints information about material properties
nodalCoords	Prints information about nodal coordinates

Figure 5.10 Properties and methods of the *Print* class

5.3 The Graphical User Interface

The user intarface of LESM is composed by a main window, where it is possible to create, view and analyze a model, and a set of auxiliary windows for getting the input data necessary to add or remove components from the model. The main window is a resizeble window designed to fit on screen resolutions of 1280 x 800 or more, and the

auxiliary windows are modal and non-resizeble. Each of these windows is an independent figure, created using the Graphical User Interface Development Environment (GUIDE) tool. This tool automatically generates a file with the code that controls the behavior of each window following the event-driven programming paradigm as described in section 4.5 (Event-Driven Programming Paradigm). The data transmission between these files is done by storing or getting the data from the root of the directory using the *setappdata* and *getappdata* commands.

The main window of the graphical interface of LESM is composed by five panels, a toolbar, a canvas and a button to run the analysis process. Figure 5.11 shows the initial configuration of this window.

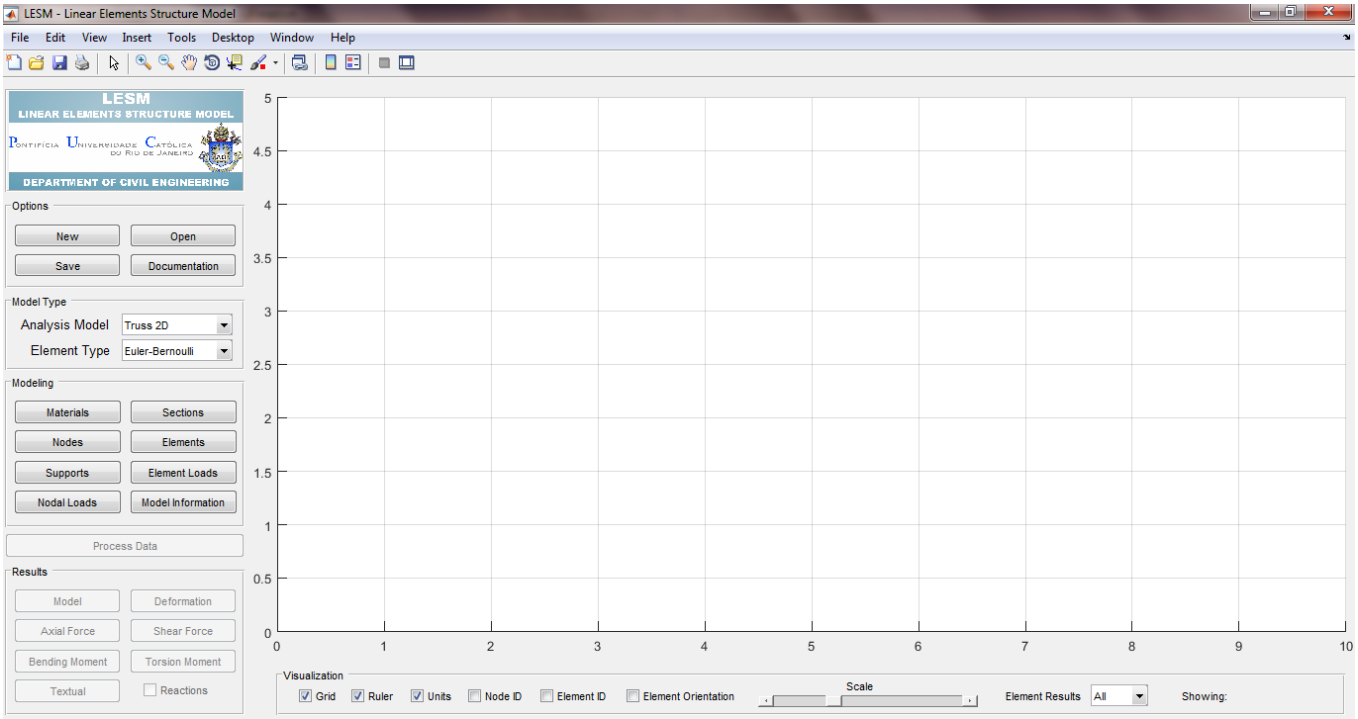

Figure 5.11 Main window of the graphical user interface of LESM

The canvas is an interface component that defines the area where the graphic system can draw geometric primitives (points, lines, shapes, etc.) to represent the structural model and the analysis results. In LESM, just like in any other MATLAB graphic application, this area is an axes system with three dimensions. The functions to control its visualization, such as zoom, pam, projection type, and camera movements, are automatically implemented.

There are three built-in toolbars that can be added to the main window: Figure Toolbar, Camera Toolbar, and Plot & Edit Toolbar. The Figure Toolbar is the only one available when the program starts and it has some basic options of visualization. The Camera Toolbar has more advanced functions of visualization and camera control. The Plot & Edit Toolbar can be used to mannually insert texts, lines and shapes to the window.

The buttons to select file-related options are in the "Options" panel. Selecting "New" will display a message to confirm if the user really wants to reset the current model. This implies in deleting all the model information and cleaning the canvas. The "Open" option displays a dialogue window to allow users to load a file containing the information of a previously saved model. There are two types of file that LESM can read, *lsm* files and neutral files, both described in section 5.3 (Data Pre-Processing). If the selected file contains valid information, the model appears on canvas, otherwise an error message shows up. The "Save" option creates a *lsm* file and saves it in the selected folder. Clicking in "Documentation" opens the published code in html format.

The “Model Type” panel has two pop-up menus, one to select the analysis model (Truss 2D, Frame 2D or Grillage) and the other to select the type of the elements i.e., the beam theory adopted for the elements (Euller-Bernoulli or Timoshenko). In LESM, a model can only have elements of the same type. When the analysis model option is changed, the camera view is adjusted to a viewpoint appropriate to the selected option. It is not possible to change these options when a model is already being built or analyzed.

The “Modelling” panel is used to create or delete model components, such as materials, cross-sections, nodes, elements, supports, prescribed displacements and loads. When clicking a button, an auxiliary GUI opens to manage the creation of objects of the selected component. These auxiliary GUIs enable users to fill only the input data fields used by the selected analysis model. When a component is created, deleted or modified, the model is automatically updated on canvas. The modelling options are always enabled so users can start building a model from the beginning or modify already existing models.

The units of input data are indicated next to each fill-in field and they cannot be changed. The units adopted are those commonly used in the daily life of engineers but they are not the units used by the data processing, so they need to be converted in the pre-processing.

In the auxiliary GUIs for managing materials and cross-sections, users can create and delete these objects as well as see a list with the properties of all created objects. All materials in LESM are considered to be homogeneous and isotropic, and the cross-sections are of a generic type. Inserting or removing objects from these lists

does not affect the model since users cannot delete a material or a cross-section that is being used by an element.

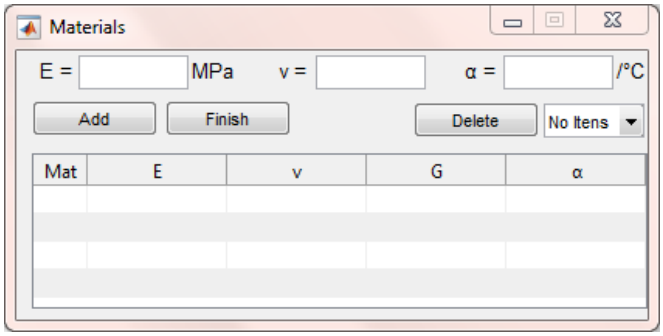

Figure 5.12 Auxiliary GUI for managing material objects

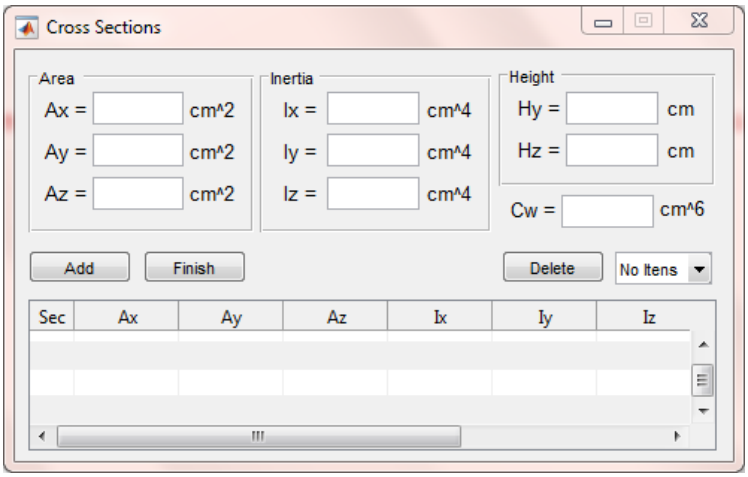

Figure 5.13 Auxiliary GUI for managing cross-section objects

The auxiliary GUI for managing nodes allows users to create nodal points by setting their coordinates. It is also possible to see a list with the coordinates of all created nodes. Users are unable to create two nodal points with the same coordinates, if it happens an error message shows up. If a node is deleted, all the connected elements are also deleted, so a warning message is displayed to confirm this action with users. The Z coordinate is not used because LESM still does not work with three-dimensional models.

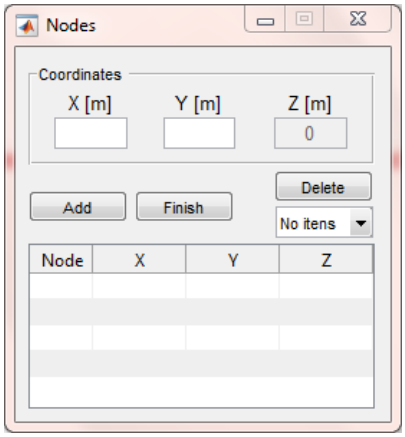

Figure 5.14 Auxiliary GUI for managing node objects

To access the auxiliary GUI that deals with elements, it is necessary to create at least one material, one cross-section, and two nodes, since information about these components are mandatory for creating an element. In this window users can create and delete element objects as well as see a list with the properties of all created elements. To create an element, the initial and final nodes cannot be the same, otherwise an error message shows up. The “Hinge 1” and the “Hinge 2” options are automatically set to “Yes” when the current analysis model is a plane truss.

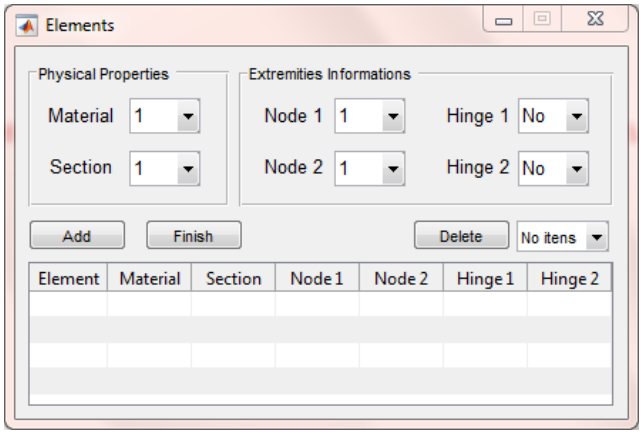

Figure 5.15 Auxiliary GUI for managing element objects

Objects of each material, cross-section, node and element are identified by a number that represents the order in which these objects were created. When an object is deleted, this order is changed, so that all objects with identification number higher than the number of the deleted object have their numbers lessened by one.

Support conditions and prescribed displacements are not created components but rather a settable property of a node. To open the auxiliary GUI for setting these properties, at least one node must be created. In this window, users can select a node to change its support conditions and prescribed displacements values. The only enabled fields to fill the prescribed displacements value are the ones with a displacement constraint in the direction of the same degree of freedom.

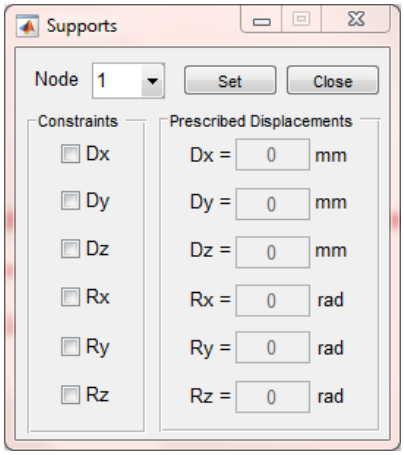

Figure 5.16 Auxiliary GUI for managing support conditions and prescribed displacements

Just like support conditions and prescribed displacements, applied nodal loads and element distributed loads are properties inherent to nodes and elements, so they are not created objects. To open the GUIs that manage these properties, there must be at least one node or element created.

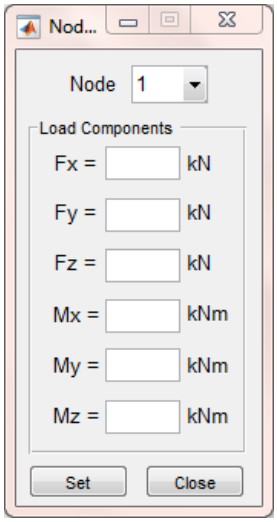

Figure 5.17 Auxiliary GUI for managing nodal loads

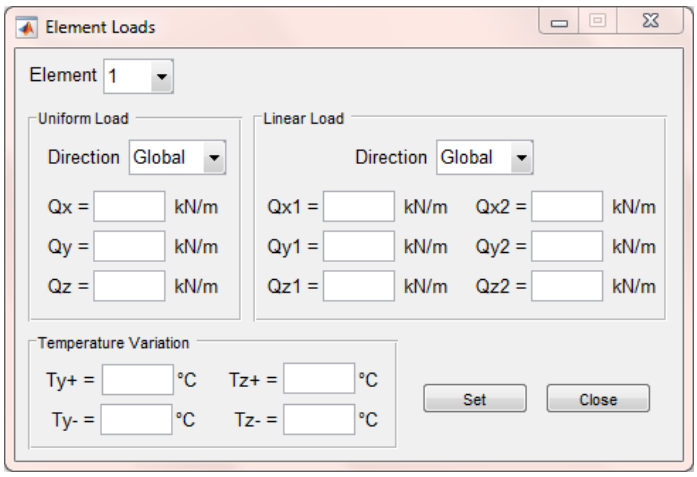

Figure 5.18 Auxiliary GUI for managing distributed loads

It is also possible to check the current model information in a text file by clicking the "Model Information" button in the "Modeling" panel.

The button for processing the data is responsible for running the code that calculates the analysis results through the direct stiffness method. These results are then set as properties of objects used in the post-processing stage to draw the graphic results or create a text file with numeric results. This button is only enabled when it is

identified that a model is being built (existence of at least one nodal point). When this button is clicked, and the data is succesfuly processed, the buttons in the “Results” panel are enabled, and the data processing button is disabled until a change is made to the model. Every time a model is loaded or modified, its data must be processed before checking the results. If the data is not succesfuly processed, which is the case of an unstable structure, a warning is issued and no changes are made.

Graphic and textual results can be checked by clicking one of the buttons in the “Results” panel. More details about results options are given in section 5.6 (Data Post-Processing). The main window also has a panel with visualization options, which are:

- Grid: Check-box to display a grid in canvas.
- Ruler: Check-box to display axes values in the side of the canvas.
- Units: Check-box to display the units of the values.
- Node IDs: Check-box to display the identification number of each node.
- Element ID: Check-box to display the identification number of each element.
- Element Orientation: Check-box to display an arrow in the middle of each element indicating its orientation from the initial node to the final node.
- Scale: Slider to adjust the scale factor of the deformed configuration and internal forces diagrams. The value of the scale factor is only informed when the deformed configuration is being displayed.
- Element Result: Pop-up menu to select the only element that will have its internal forces diagrams displayed.
- Showing: Text to indicate what is being shown in canvas.

5.4 Data Pre-Processing

This stage consists in providing input data with all the necessary information about a structural model and preparing it to be used in the processing stage. This information can be provided by the process of modeling or by loading a file containing the model full description in a format that LESM can read and interpret.

When users load a file, an auxiliary function, called *openFile*, is executed to read the model information and create objects to store this information in their properties. There are two file formats that this function can read: *lsm* file and Neutral File. These formats are very similar, with the *lsm* file being an adaptation of the Neutral File. After loading a file, the model immediately appears on canvas, but its data is not yet processed. When a model is saved, its data is stored in a *lsm* file.

The Neutral File is actually a file format designed with the purpose of containing all information needed for finite element analysis programs. It is easy to read, has a very simple structure, contains no automatic generation or implicit rule, and provides a simple way for programs to skip information that is not needed for their specific function. Both the Neutral File and the *lsm* file specify a 3D model. Therefore, LESM always reads and stores information about a three-dimensional model even if, depending on the analysis model, some of these data are not used in the analysis process. An object of an *Anm* subclass defines what information is used in the data processing. More about the Neutral File format can be found at http://www.tecgraf.puc-rio.br/neutralfile.

The modeling process allows users to modify a loaded model or to build one from the beginning by gradually providing information about materials, cross-sections, nodes, elements, support conditions, and loads. This information is provided through the input data of the auxiliary GUIs.

To avoid mistakes during the modeling process, the program needs to control the actions of the users. In LESM, one of the mechanisms to make this control is to disable the input data fields of the information that is not used in the selected analysis model, avoiding users to get confused. As mentioned, even if this unnecessary information is defined by users, the program would still provide the correct results because an object of the *Anm* class would access only the right properties. Figure 5.19 brings a list of properties that are enabled to be set in each analysis model.

Another implementation that helps users not to make mistakes during modeling was the development of a method called *fictRotConstraint*. This method inserts or removes a fictitious rotation constraint in nodes with more than one element connected and which all of them are hinged at this end. This can be a common mistake when trying to release the rotation between two or more elements, and causes the model to be unstable. This method is called immediately before the data processing, to insert the fictitious constraints, if any, and after, to remove.

Property	Truss_2D	Frame_2D	Grillage
Cross Section Area	Ax	Ax Ay	Ax Az
Cross Section Inertia	-	Iz	Ix Iy
Cross Section Height	Hy	Hy	Hz
Constraints	Dx Dy	Dx Dy Rz	Dz Rx Ry
Prescribed Displacements	Dx Dy	Dx Dy Rz	Dz Rx Ry
Nodal Loads	Fx Fy	Fx Fy Mz	Fz Mx My
Element Uniform Load	Qx Qy	Qx Qy	Qz
Element Linear Load	Qx1 Qy1 Qx2 Qy2	Qx1 Qy1 Qx2 Qy2	Qz1 Qz2
Element Temperature Variation	Ty+ Ty-	Ty+ Ty-	Tz+ Tz-

Figure 5.19 Settable properties of each analysis model

5.5 Data Processing

After loading, building or modifying a structural model, the “Process Data” button is enabled. When this button is clicked, the method “process” of the *Drv* class is called to execute the data processing and to enable the visualization of the graphic and textual results. This stage consists of a group of functions implemented to use the data provided by the pre-processing stage for calculating the analysis results of a structural model. The processing stage of LESM is completely based on the OOP paradigm, so these functions are actually class methods and the provided data are the properties of objects created in the data pre-processing. After this stage is executed, the analysis results are also stored in the properties of these objects that need to be accessed from the objects of the post-processing classes.

The algorithm to calculate the analysis results follows the direct stiffness method. This is a matrix method that discretizes the model in nodal displacements and uses the elements’ stiffness relations for calculating the unknown displacements of the nodes, the support reactions, and the internal forces at element ends.

In summary, the target problem is to solve the system of equilibrium equations of the structural model given in equation 5.1, where {F} is the global vector of forces, [K] is the global stiffness matrix, and {D} is the global vector of nodal displacements. The unknowns of this system are the values of nodal displacements in the direction of free degrees of freedom, since displacements in the restricted directions have null or prescribed values, and the values of forces acting in the direction of fixed degrees of freedom that are the support reactions. The number of equations in this system is equivalent to the total number of degrees of freedom of the model.

$$\{F\} = [K]\{D\} \tag{5.1}$$

After solving this system, it is necessary to compute the internal forces at the ends of the elements. This is done by combining the result of the global analysis, which takes into account only the nodal displacements, with the result from the local analysis of loaded elements.

The method can be divided into three parts, explained in the following sections: The assembly of the equation system, the resolution of this system, and the computation of internal forces.

5.5.1 Assembly of the Equation System

In this first part of the method, the terms of the equation system must be assembled in such a way that its resolution can be partitioned. The flowchart in Figure 5.20 indicates the order of the methods called to assemble the global stiffness matrix, the global displacements vector, and the global forces vector. The boxes of this flowchart give the name and the class of the methods, not specifying subclasses. The methods of the *Elem* and *Lelem* classes that deal with the flexural behavior are being generically treated by not specifying the plane where the bending occurs.

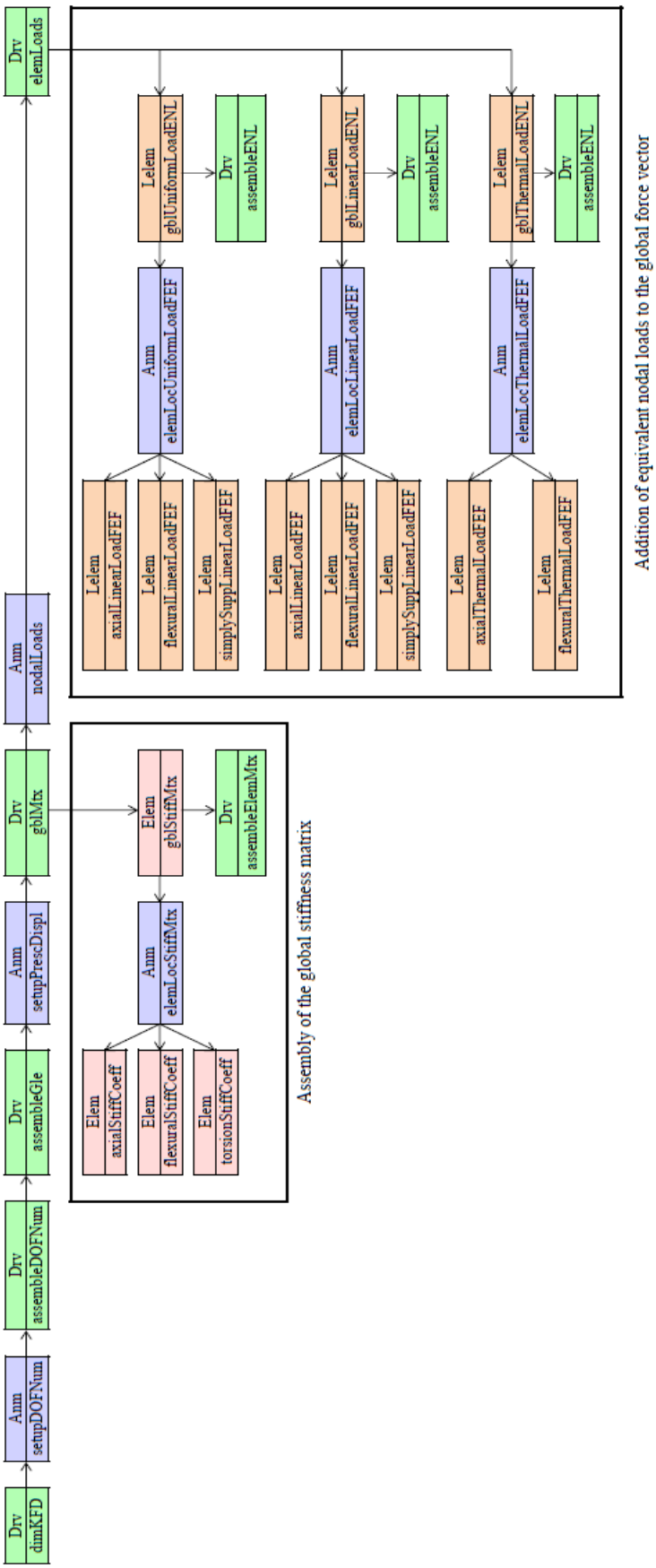

Figure 5.20 Flowchart of the methods to assemble the equation system

5.5.1.1 Initialization of Auxiliary Variables

The first step of the analysis process is to initialize and dimension the global stiffness matrix, the global displacements vector, and the global forces vector. This is done in the "dimKFD" method by creating a square matrix and two column vectors, all filled with zeros and of the size of the number of equations in the system.

In order to partition the equation system, the equations related to free degrees of freedom must be numbered first, before the equations of fixed degrees of freedom. This numbering order is set in the "setupDOFNum" method. After that, two auxiliary variables must be created to store the equation number of the degrees of freedom of each node and element. These variables are the ID matrix and the gather vector of the elements.

The ID matrix has the number of columns equal to the total number of nodes in the model, and the number of rows equal to the number of degrees of freedom per node. The term $ID_{(ij)}$ gives the equation number of the degree of freedom "i" of node "j". This matrix is assembled in the "assembleDOFNum" method.

The gather vector is a column vector of the size of the number of degrees of freedom of an element, which is equal to the sum of the degrees of freedom of its initial and final nodes. This vector gives the equation number of the degrees of freedom of each element and it is set as a property of the objects of the Elem class. The gather vectors of the elements are assembled in the "assembleGle" method.

5.5.1.2 Assembly of the Global Displacements Vector

The global displacements vector stores the values of nodal displacements in the global system. The assembly order of this vector follows the numbering order of the equations so that displacements of fixed degrees of freedom come in the end of the vector.

The first part of this vector corresponds to the unknowns of the equation system, while the second part stores the known support displacement values. The displacements in the direction of fixed degrees of freedom are usually zero, unless a value had been prescribed. The "setupPresDispl" method stores the prescribed displacements values in the global displacements vector.

5.5.1.3 Assembly of the Global Stiffness Matrix

The global stiffness matrix of structural models is the matrix that correlates the global displacements vector and the global forces vector. This matrix is assembled through a direct sum of the stiffness matrices of each element, hence the name of the method. The "gblMtx" method is responsible for this task.

The stiffness matrices of linear elements were presented in section 3.3 (Stiffness Matrices) in the local system using the terms of the stiffness coefficient matrices, given in Appendix A (Element Local Stiffness Coefficients). The stiffness coefficient matrices are generated in the methods *axialStiffCoeff*, *flexuralStiffCoeff_XY*, *flexuralStiffCoeff_XZ* and *torsionStiffCoeff*. The assembly of the stiffness matrix of each element is done in the "elemLocStiffMtx" method by allocating the terms of the stiffness coefficient matrices in the correct position of the element stiffness matrix.

The stiffness matrix of each element is then rotated to the global system in the "gblStiffMtx" method using rotation transformation matrix of the element, as shown in equation 3.4. The "gblMtx" method gets these rotated matrices and calls the "assembleElemMtx" method to insert them in the correct positions of the global stiffness matrix.

To add the contribution of the stiffness matrix of an element, in the global system, to the global stiffness matrix, the gather vector is used to associate the element degrees of freedom to the corresponding equation number in the global matrix.

5.5.1.4 Assembly of the Global Forces Vector

The global forces vector stores the values of the forces that act on nodes in the direction of the degrees of freedom in the global system. The assembly order of this vector follows the numbering order of the equations so that the unknown forces related to fixed degrees of freedom (support reactions) come in the end of the vector, while the known force values come in the beginning.

The value of the forces can be obtained by combining the contributions coming from applied nodal loads and equivalent nodal loads of loaded elements. Initially, it is assumed that the unknown terms of the vector store loads applied directly to fixed degrees of freedom.

The "nodalLoads" method is responsible for adding the contribution of applied nodal loads to any term of the global forces vector, including terms that correspond to fixed degrees of freedom. The ID matrix is used to associate the degrees of freedom of the nodes to the corresponding equation number in the global forces vector.

The equivalent nodal loads are the fixed end forces of loaded elements, explained in section 3.4 (Fixed End Forces), in the opposite direction. The "elemLoads" method adds the equivalent nodal loads from each loaded element, including distributed loads and thermal load, to the global forces vector.

The vectors of fixed end forces of linear elements are given in the local system, using terms of the decoupled fixed end forces vectors for each individual effect. These decoupled vectors are given in Appendix A (Fixed End Forces from Linear Distributed Loads) and Appendix B (Fixed End Forces from Temperature Variation), and generated in the methods *axialLinearLoadFEF*, *axialThermalLoadFEF*, *flexuralLinearLoadFEF_XY*, *flexuralLinearLoadFEF_XZ*, *FlexuralThermalLoad_XZ*, *flexuralThermalLoadFEF_XY*, and *simplySuppLinearLoadFEF*. The assembly of the fixed end forces vectors of each element is done in the methods *elemLocUniformLoadFEF*, *elemLocLinearLoadFEF* and *elemLocThermalLoadFEF* by allocating the terms of the decoupled vectors in the correct position of the element fixed end forces vector.

To calculate the vector of equivalent nodal loads of an element, its fixed end forces vector is then rotated to the global system and multiplied by -1 to invert the direction of the forces. This is done in the methods *gblUniformLoadENL*, *gblLinearLoadENL*, and *gblThermalLoadENL* using the rotation transformation matrix as shown in equation 3.11. The "elemLoads" method gets these rotated vectors and calls the "assembleENL" method to insert them in the correct positions of the global forces vector, including those that correspond to fixed degrees of freedom. The gather vectors are used to associate the degrees of freedom of the elements to the corresponding equation number in the global forces vector.

5.5.2 Resolution of the Equation System

After assembling the system of equilibrium equations, it must be solved for the unknown values of nodal displacements and support reactions. The numbering order of the degrees of freedom, considering the free before the fixed, makes it possible to divide the equation system into equations 5.2 and 5.3. The partition of the global stiffness matrix, global displacements vector, and global forces vector is described below. The subscript "f" refers to free and "c" refers to constrained.

- $\{D_f\}$→ Vector of nodal displacements in the directions of free degrees of freedom (unknown)
- $\{F_f\}$→ Vector of forces in the directions of free degrees of freedom (known)
- $\{D_c\}$→ Vector of nodal displacements in the directions of fixed degrees of freedom (known)
- $\{F_c\}$→ Vector of forces in the directions of fixed degrees of freedom (unknown)

$$[K_{ff}]\{D_f\} + [K_{fc}]\{D_c\} = \{F_f\} \tag{5.2}$$

$$[K_{cf}]\{D_f\} + [K_{cc}]\{D_c\} = \{F_c\} \tag{5.3}$$

The manipulation of equation 5.2 results in equation 5.4, where D_f is the only unknown.

$$[K_{ff}]\{D_f\} = \{F_f\} - [K_{fc}]\{D_c\} \tag{5.4}$$

With D_f determined, it is possible to calculate F_c using equation 5.3 and adding the global forces of fixed degrees of freedom in the opposite direction.

$$\{F_c\} = [K_{cf}]\{D_f\} + [K_{cc}]\{D_c\} - \{F_c\} \tag{5.5}$$

The partition of the equation system and its resolution is done in the "solveEqnSystem" method of the *Drv* class. The stability of the model is also verified in this method by checking if the submatrix K_{ff} is singular or not. If it is, the model is unstable and the data processing is interrupt with an error message.

5.5.3 Computation of Internal Forces at Element Ends

The internal forces of the elements are initially calculated only at their ends. This calculation must take into account the effects of the global and local analysis of the structural model. The value of these end forces is presented in textual format. For generating the graphic results, it is necessary to calculate the internal forces along the length of the elements, based on the value of their end forces. The generation of internal forces diagrams is part of the post-processing stage.

The global analysis gives the internal forces of elements resulting from the effect of nodal displacements only. The value of these displacements are calculated when the system of equilibrium equations is solved. The local analysis of elements gives the effect of internal loads along the length of the elements and it must be added to the effect of nodal displacements.

All the process of computing internal forces is done in the "elemIntForce" method as indicated in the flowchart of Figure 5.21. This flowchart, just like the one in Figure 5.20, does not specify the plane where the bending occurs for methods related to flexural behavior.

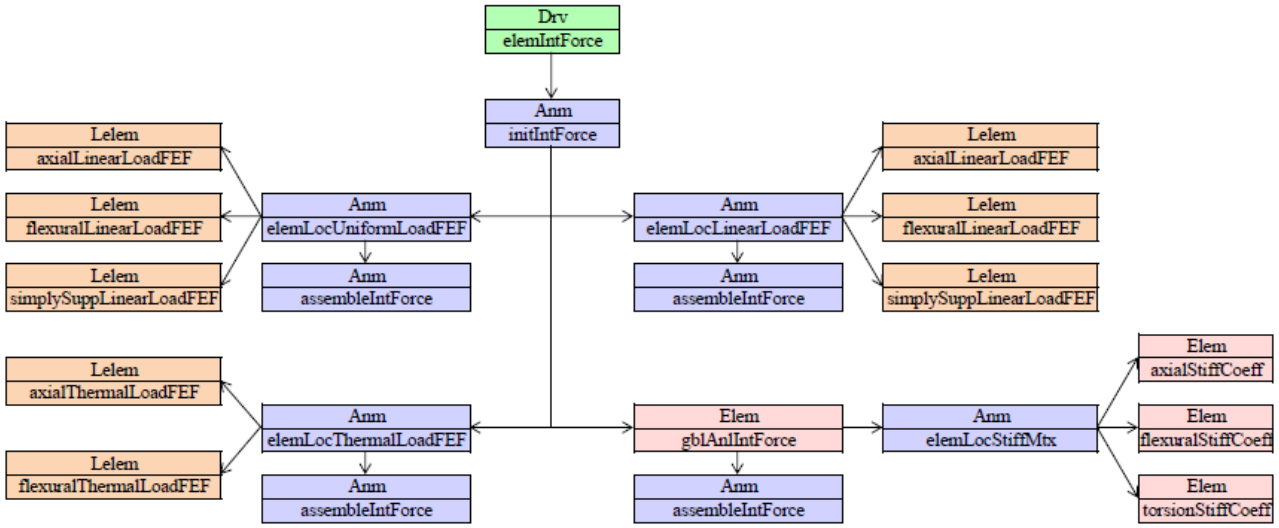

Figure 5.21 Flowchart of the methods to compute internal forces at element ends

Initially, the vectors of internal forces of each element are initialized with null values in the "initIntForce" method. Then, the "elemIntForce" method adds the contribution of global analysis to these empty vectors by calling the "gblAnlIntForce" method and the "assembleIntForce" method. The first uses the gather vector of an element to get its vector of nodal displacements, convert it to the local system using the rotation transformation matrix, and multiplies it by the element stiffness matrix that must be assembled one more time. The result of this product is the vector of generalized forces, which are the forces applied at the ends of an element to keep it in the deformed configuration. This vector is then added to the vectors of internal forces in the "assembleIntForce" method.

After considering the effect of the global analysis, the "elemIntForce" method adds the contribution of each type of element load by getting the vector of fixed end forces of all loaded elements and calling the "assembleIntForce" method to add them to the vectors of internal forces. The vectors of fixed end forces must be assembled once again, as it is done in the assembly of the global forces vector (section 5.5.1.4).

The sign convention of the calculated internal forces is not yet the same as that established in chapter 2 (Linear Element Models). The way it is, the internal forces are positive when acting in the positive direction of the corresponding degree of freedom. This sign convention is kept when printing the textual results, but it needs to be converted to the usual convention in the post-processing stage to display the diagrams.

It is clear that this algorithm is not the most efficient implementation, since the stiffness matrix of all elements and the vector of fixed end forces of all loaded elements are being calculated and assembled twice along the data processing. The reason for this is that this program values the didactics and understanding of the code by its users, and implementing the code this way is better to visualize how the direct stiffness method works.

5.6 Data Post-Processing

Once the results of the structural analysis have been computed in the processing stage, the data post-processing becomes responsible for making these results available to users by graphically displaying or printing them. This section is dedicated to explain the considerations and the development of the code for exposing the analysis results. The results options of LESM are exhibited in the “Results” panel of Figure 5.22. The “Model” button in this panel is not considered as a result option, but rather an option to show the structural model.

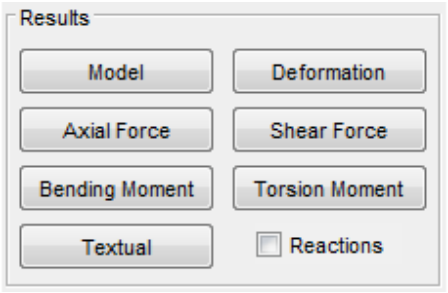

Figure 5.22 Panel of results options

The graphic components of this panel are enabled only when the data of the current model has been successfully processed. Only the buttons that provide the internal forces supported by the selected analysis model are enabled. When a button or the "Reactions" checkbox is clicked, a method of a post-processing class is called by the callback function that controls the behavior of the main window. These classes are discussed in sections 5.2.8 (The *Draw* Class) and 5.2.9 (The *Print* Class).

The interpretation of the internal forces diagrams depends on the convention adopted to determine the lower and upper face of an element. In horizontal and inclined elements of plane truss and plane frame models, the lower face is on the side of the element where the vertical component of its orthogonal vector points in the negative direction of the global Y axis. In vertical elements, the lower face is on the right side. In grillage models, the lower face is on the side of the element where its orthogonal vector points in the negative direction of the global Z axis.

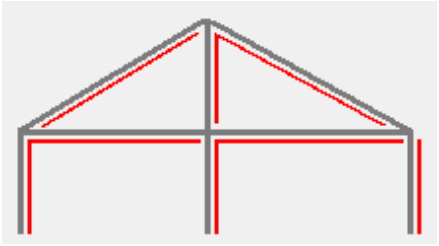

Figure 5.23 Lower face of elements in plane truss and plane frame models

5.6.1 Deformed Configuration

The algorithm of the “deformConfig” method for drawing the deformed configuration of plane truss models is different from the algorithm for drawing plane frame and grillage models. In the case of plane trusses, all elements are hinged at their ends and the internal effect of element loads are ignored since the loads action along an element is statically transferred as concentrated forces to the element end nodes. Because of this, this analysis model accepts only constant axial internal forces and the elements always remain straight in the deformed configuration. The only information needed for drawing the deformed configuration of a truss model is the coordinates of the displaced nodes, which is the sum of the original coordinates with the nodal displacements given by the global displacements vector. The deformed configuration is drawn simply by connecting the displaced nodal coordinates.

In the case of plane frame and grillage models, not all elements are hinged at their ends and the internal effects of element loads must be considered when drawing the structure deformed configuration. Because of this, elements may have an internal deformation resulting from nodal displacements (global analysis) and from the internal loads (local analysis). To graphically represent elements deformation, they are discretized in several nodes where the internal displacements are calculated using the shape functions (result from global analysis) and the expressions for internal displacements in loaded elements (result from local analysis). The displaced coordinates of these internal nodes are then connected to draw the deformed configuration of the element.

The shape functions are used to calculate the displacements of each node created inside an element as a result of the displacements at the element ends. This is done by assembling the shape functions matrix with the values of the shape functions expressions, given in Appendix D (Shape Functions). Equation 3.14 is then applied by multiplying the shape functions matrix by the vector of nodal displacements at the element ends, in the local system. The result is a vector with the axial and transversal displacement components of the internal node.

The expressions for internal displacements in loaded elements are used to calculate the displacements of each node inside an element as a result of the local analysis. The expressions that describe the axial and transversal displacement components are given in Appendix E (Internal Displacement from Linearly Distributed Load) and Appendix F (Internal Displacement from Temperature Variation). To determine the total displacement of an internal node, the displacement components resulting from the global and local analysis are then combined.

The components of the total displacement of each internal node is added to their original coordinate to graphically display the element deformed configuration by connecting the points. The initial scale factor is calculated in the "scaleFactor" method, and can be changed at any time using the slider in the "Visualization" panel. Its value is indicated in the same panel. This factor multiplies the total displacements of internal nodes before adding this value to the original coordinates.

5.6.2 Axial Force Diagram

The value of the axial force on elements of plane truss models is always constant, so instead of indicating it in a diagram, this value is plotted next to each element. The reason for this is that the visualization of the results becomes clearer. This constant value is that calculated at the ends of the element, but with the sign convention converted to the usual convention established in chapter 2 (Linear Element Models).

In plane frame models, the value of the internal axial force is constant only on elements with no distributed axial loads. In this case, the diagram is drawn by connecting the initial and final values. If an element is loaded, its axial force diagram is drawn by discretizing it in several nodes and calculating the force value in the position of each of these nodes. These values are then multiplied by a scale factor and connected to draw the diagram.

The function that gives the value of the axial force in any position of loaded elements, N(x), can be obtained from the differential equation 5.6.

$$\frac{dN(x)}{dx} = -p(x) \tag{5.6}$$

The function that describes the axial load distribution on an element, p(x), is computed combining both its uniform and linear loads. Assuming a generic load distribution for this combination, in the form of equation 5.7, the internal axial force is expressed by equation 5.8.

$$p(x) = Ax + B \tag{5.7}$$

$$N(x) = -A\frac{x^2}{2} - Bx + C \tag{5.8}$$

The constant “C” is calculated using the boundary condition of axial force at element initial end. The axial force diagram is displayed with positive values plotted on the upper side of the element. The initial scale factor is calculated in the “scaleFactor” method, and can be changed at any time using the slider in the “Visualization” panel, but its value is not shown for any internal force diagram. Grillage models do not support internal axial force.

5.6.3 Shear Force Diagram

In plane frame and grillage models, the shear force diagram is drawn the same way the axial force diagram is, but in this case, when an element has a distributed load the transversal component is considered and the shear force is calculated in the position of each internal node. Truss models do not support internal shear force.

The function that gives the value of the shear force in any position of loaded elements, V(x), can be obtained from the differential equation 5.9.

$$\frac{dV(x)}{dx} = -q(x) \tag{5.9}$$

The function that describes the transversal load distribution on an element, q*(x)*, and the shear force equation are:

$$q(x) = Ax + B \tag{5.10}$$

$$V(x) = -A\frac{x^2}{2} - Bx + C \tag{5.11}$$

The constant “C” is calculated using the boundary condition of shear force at element initial end. The shear force diagram is displayed with positive values plotted on the upper side of the element.

5.6.4 Bending Moment Diagram

Just like the axial and shear force diagrams, the bending moment diagram in plane frame and grillage models is drawn by discretizing loaded elements in several internal nodal points. Only the transversal component of distributed loads is considered and the function that describes its distribution is the same of equation 5.10. The function that gives the value of the bending moment in any position of loaded elements, M(x), can be obtained from the differential equation 5.12, resulting in equation 5.13 when considering a generic linear load distribution.

$$\frac{d^2M(x)}{dx^2} = -q(x) \tag{5.12}$$

$$M(x) = -A\frac{x^3}{6} - B\frac{x^2}{2} + Cx + D \tag{5.13}$$

The constant "C" is calculated using the boundary condition of shear force at element initial end in equation 5.11, and the constant "D" using the boundary condition of bending moment at the initial end, in equation 5.13. The bending moment diagram is displayed with positive values plotted on the tension side of the element. Truss models do not support internal bending moment.

5.6.5 Torsion Moment Diagram

In grillage models, the value of torsion moment on elements is always constant since distributed torsion moment is not a load type considered in LESM. So, instead of indicating the values in a diagram, they are plotted next to each element. This constant value is that calculated at the ends of the element, but with the sign convention converted to the usual convention established in chapter 2 (Linear Element Models). Plane truss and plane frame models do not support internal torsion moment.

5.6.6 Other Result Options

When the callback function that controls the “Textual” button behavior is triggered, it creates a text file and calls the methods of the *Print* class responsible for printing the results. The provided results are nodal displacements, support reactions, and internal forces at element ends in the original sign convention.

The “Reactions” checkbox, when activated, displays the support reactions values next to each support.

6 Conclusions

The development of LESM proved that the computational implementation of the direct stiffness method for structural analysis is a task that makes the learning of this method much easier. When writing a code with the algorithm of this method, or any other method whose manual resolution is not feasible, the procedures become clearer to be understood. The understanding of the data processing of a structural analysis program is then a great complement to the theory presented in the courses of matrix analysis of structures.

Because LESM is a simple graphic program, it was also possible to have an overview of all stages of the development of more complex structural analysis programs, such as those used commercially by companies or universities. Many of the concepts presented in this work for implementing the data processing, pre-processing and post-processing stages are generic for developing programs of any field of study.

The use of the object-oriented programming paradigm, although not trivial for some, has proved to be a very clear way to show the code of the program, especially when using a high-level language such as MATLAB. This also allows the program to be expanded much more easily than if it had been written in a procedural programming paradigm. Future versions of LESM may include three-dimensional models, semi-rigid support conditions and the consideration of second order effects.

References

1. Martha, L.F., “Análise de Estruturas: Conceitos e Métodos Básicos”, Elsevier, 2010.
2. Martha, L.F., “Análise Matricial de Estruturas: Aplicada a Modelos Lineares”, Elsevier, 2016.
3. Soriano, H. L., “Análise de Estruturas: Formulação Matricial e Implementação Computacional”, Ciência Moderna, 2005.
4. Hibeler, R. C., “Mechanics of Materials”, 7th edition, Pearson, 2008.
5. Register, A. H., “A Guide to MATLAB Object-Oriented Programming”, Chapman & Hall/CRC, 2007.
6. Attaway, S., “Matlab: a Practical Introduction to Programming and Problem Solving”, 4th edition, Butterworth-Heinemann/Elsevier, 2016.
7. Chapman, S. J., “MATLAB Programming for Engineers”, 5th edition, Chegg, 2016.
8. Santana, M.. V. B., “Desenvolvimento de Sistema Computacional via MATLAB/GUI (Graphical User Interface) para Análise Geometricamente Não Linear de Estruturas”, UFOP, 2015.
9. Gattas, M., “Introdução à Computação Gráfica”, PUC-Rio, 2013.

Appendix A Local Stiffness Coefficients of Linear Elements

A.1 Axial Stiffness Coefficients

$$[\mathrm{kea}] = \begin{bmatrix} \frac{EA}{L} & -\frac{EA}{L} \\ -\frac{EA}{L} & \frac{EA}{L} \end{bmatrix} \tag{A.1}$$

A.2 Torsional Stiffness Coefficients

Element with continuous ends:

$$[\mathrm{ket}] = \begin{bmatrix} \frac{GJ}{L} & -\frac{GJ}{L} \\ -\frac{GJ}{L} & \frac{GJ}{L} \end{bmatrix} \tag{A.2}$$

Element with any hinged end:

$$[\mathrm{ket}] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{A.3}$$

A.3 Flexural Stiffness Coefficients

Parameters of Timoshenko beam theory (for Euler-Bernoulli beam theory, $\Omega = 0$):

$$\Omega = \frac{EI}{G A_s L^2} \tag{A.4}$$

$$\lambda = 1 + 3\Omega \tag{A.5}$$

$$\mu = 1 + 12\,\Omega \tag{A.6}$$

Element with continuous ends:

$$[\text{kef}]_{xy} = \begin{bmatrix} \frac{12EI}{\mu L^3} & \frac{6EI}{\mu L^2} & -\frac{12EI}{\mu L^3} & \frac{6EI}{\mu L^2} \\ \frac{6EI}{\mu L^2} & \frac{\lambda 4EI}{\mu L} & -\frac{6EI}{\mu L^2} & \frac{\gamma 2EI}{\mu L} \\ -\frac{12EI}{\mu L^3} & -\frac{6EI}{\mu L^2} & \frac{12EI}{\mu L^3} & -\frac{6EI}{\mu L^2} \\ \frac{6EI}{\mu L^2} & \frac{\gamma 2EI}{\mu L} & -\frac{6EI}{\mu L^2} & \frac{\lambda 4EI}{\mu L} \end{bmatrix} \quad \text{(A.7)}$$

$$[\text{kef}]_{xz} = \begin{bmatrix} \frac{12EI}{\mu L^3} & -\frac{6EI}{\mu L^2} & -\frac{-12EI}{\mu L^3} & -\frac{6EI}{\mu L^2} \\ -\frac{6EI}{\mu L^2} & \frac{\lambda 4EI}{\mu L} & \frac{6EI}{\mu L^2} & \frac{\gamma 2EI}{\mu L} \\ -\frac{12EI}{\mu L^3} & \frac{6EI}{\mu L^2} & \frac{12EI}{\mu L^3} & \frac{6EI}{\mu L^2} \\ -\frac{6EI}{\mu L^2} & \frac{\gamma 2EI}{\mu L} & \frac{6EI}{\mu L^2} & \frac{\lambda 4EI}{\mu L} \end{bmatrix} \quad \text{(A.8)}$$

Element with initial end hinged:

$$[\text{kef}]_{xy} = \begin{bmatrix} \frac{3EI}{\lambda L^3} & 0 & -\frac{3EI}{\lambda L^3} & \frac{3EI}{\lambda L^2} \\ 0 & 0 & 0 & 0 \\ -\frac{3EI}{\lambda L^3} & 0 & \frac{3EI}{\lambda L^3} & -\frac{3EI}{\lambda L^2} \\ \frac{3EI}{\lambda L^2} & 0 & -\frac{3EI}{\lambda L^2} & \frac{3EI}{\lambda L} \end{bmatrix} \quad \text{(A.9)}$$

$$[\text{kef}]_{xz} = \begin{bmatrix} \frac{3EI}{\lambda L^3} & 0 & -\frac{3EI}{\lambda L^3} & -\frac{3EI}{\lambda L^2} \\ 0 & 0 & 0 & 0 \\ -\frac{3EI}{\lambda L^3} & 0 & \frac{3EI}{\lambda L^3} & \frac{3EI}{\lambda L^2} \\ -\frac{3EI}{\lambda L^2} & 0 & \frac{3EI}{\lambda L^2} & \frac{3EI}{\lambda L} \end{bmatrix} \quad \text{(A.10)}$$

Element with final end hinged:

$$[\text{kef}]_{xy} = \begin{bmatrix} \frac{3EI}{\lambda L^3} & \frac{3EI}{\lambda L^2} & -\frac{3EI}{\lambda L^3} & 0 \\ \frac{3EI}{\lambda L^2} & \frac{3EI}{\lambda L} & -\frac{3EI}{\lambda L^2} & 0 \\ -\frac{3EI}{\lambda L^3} & -\frac{3EI}{\lambda L^2} & \frac{3EI}{\lambda L^3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{A.11}$$

$$[\text{kef}]_{xz} = \begin{bmatrix} \frac{3EI}{\lambda L^3} & -\frac{3EI}{\lambda L^2} & -\frac{3EI}{\lambda L^3} & 0 \\ -\frac{3EI}{\lambda L^2} & \frac{3EI}{\lambda L} & \frac{3EI}{\lambda L^2} & 0 \\ -\frac{3EI}{\lambda L^3} & \frac{3EI}{\lambda L^2} & \frac{3EI}{\lambda L^3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{A.12}$$

Element with hinged ends:

$$[\text{kef}] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{A.13}$$

Appendix B Fixed End Forces from Linearly Distributed Load

B.1 Fixed End Forces from Axial Distributed Load

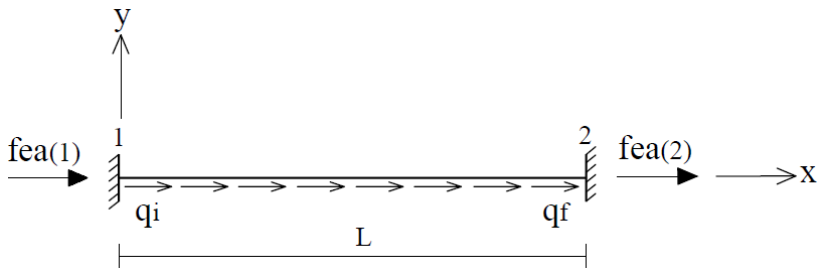

Figure B.1 Fixed end forces from axial distributed load in the XY plane

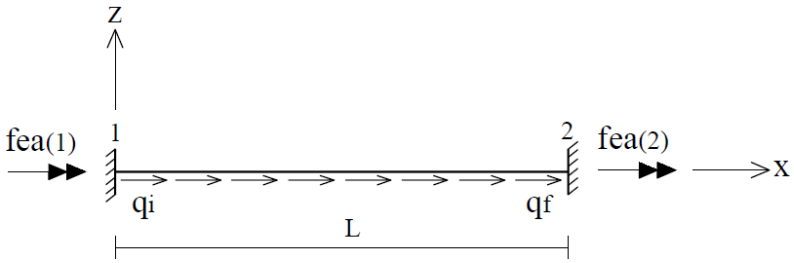

Figure B.2 Fixed end forces from axial distributed load in the XZ plane

$$[\text{fea}]_{xy} = \begin{bmatrix} -\left(\frac{q_i L}{3} + \frac{q_f L}{6}\right) \\ -\left(\frac{q_i L}{6} + \frac{q_f L}{3}\right) \end{bmatrix} \tag{B.1}$$

$$[\text{fea}]_{xz} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{B.2}$$

B.2 Fixed End Forces from Transversal Distributed Load

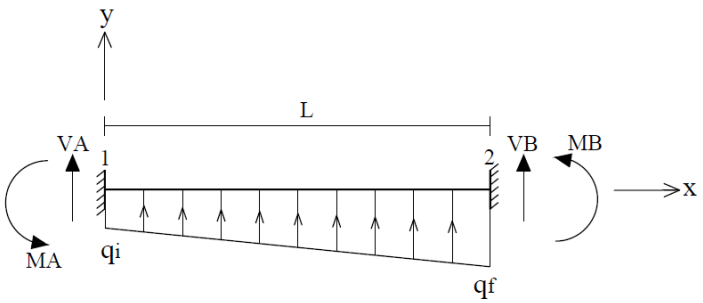

Figure B.3 Fixed end forces from transversal distributed load in the XY plane

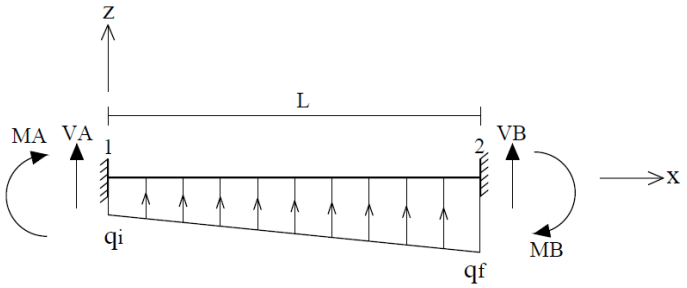

Figure B.4 Fixed end forces from transversal distributed load in the XZ plane

$$q_0 = q_i \tag{B.3}$$

$$q_1 = q_f - q_i \tag{B.4}$$

Parameters of Timoshenko beam theory (for Euler-Bernoulli beam theory, $\Omega = 0$):

$$\Omega = \frac{EI}{G A_s L^2} \tag{B.5}$$

$$\gamma = 1 - 6\Omega \tag{B.6}$$

$$\lambda = 1 + 3\Omega \tag{B.7}$$

$$\mu = 1 + 12\,\Omega \tag{B.8}$$

$$r = 1 + \frac{40\Omega}{3} \tag{B.9}$$

$$s = 1 + 15\Omega \tag{B.10}$$

$$t = 1 + \frac{80\Omega}{7} \tag{B.11}$$

$$u = 1 + 10\Omega \tag{B.12}$$

Fixed end forces values for an element with continuous ends:

$$\mathrm{VA} = -\left(\frac{q_0 L}{2} + \frac{3q_1 L\, r}{20\mu}\right) \tag{B.13}$$

$$\mathrm{MA_{xy}} = -\left(\frac{q_0 L^2}{12} + \frac{q_1 L^2}{30\mu}\right) \tag{B.14}$$

$$\mathrm{MA_{xz}} = \left(\frac{q_0 L^2}{12} + \frac{q_1 L^2}{30\mu}\right) \tag{B.15}$$

$$\mathrm{VB} = -\left(\frac{q_0 L}{2} + \frac{7q_1 L\, t}{20\mu}\right) \tag{B.16}$$

$$\mathrm{MB_{xy}} = \left(\frac{q_0 L^2}{12} + \frac{q_1 L^2 u}{20\mu}\right) \tag{B.17}$$

$$\mathrm{MB_{xz}} = -\left(\frac{q_0 L^2}{12} + \frac{q_1 L^2 u}{20\mu}\right) \tag{B.18}$$

Transmission coefficient of moments:

$$\mathrm{tAB} = \mathrm{tBA} = \frac{\gamma}{2\lambda} \tag{B.19}$$

Element with continuous ends:

$$[\text{fef}] = \begin{bmatrix} \text{VA} \\ \text{MA} \\ \text{VB} \\ \text{MB} \end{bmatrix} \tag{B.20}$$

Element with initial end hinged:

$$[\text{fef}]_{\text{xy}} = \begin{bmatrix} \text{VA} - \dfrac{\text{MA}(1 + \text{tAB})}{L} \\ 0 \\ \text{V}B + \dfrac{\text{MA}(1 + \text{tAB})}{L} \\ \text{MB} - \text{MA} \times \text{tAB} \end{bmatrix} \tag{B.21}$$

$$[\text{fef}]_{\text{xz}} = \begin{bmatrix} \text{VA} + \dfrac{\text{MA}(1 + \text{tAB})}{L} \\ 0 \\ \text{V}B - \dfrac{\text{MA}(1 + \text{tAB})}{L} \\ \text{MB} - \text{MA} \times \text{tAB} \end{bmatrix} \tag{B.22}$$

Element with final end hinged:

$$[\text{fef}]_{\text{xy}} = \begin{bmatrix} \text{VA} - \dfrac{\text{MB}(1 + \text{tBA})}{L} \\ \text{MA} - \text{MB} \times \text{tBA} \\ \text{V}B + \dfrac{\text{MB}(1 + \text{tBA})}{L} \\ 0 \end{bmatrix} \tag{B.23}$$

$$[\text{fef}]_{\text{xz}} = \begin{bmatrix} \text{VA} + \dfrac{\text{MB}(1 + \text{tBA})}{L} \\ \text{MA} - \text{MB} \times \text{tBA} \\ \text{V}B - \dfrac{\text{MB}(1 + \text{tBA})}{L} \\ 0 \end{bmatrix} \tag{B.24}$$

Element with hinged ends:

$$[\text{fef}] = \begin{bmatrix} -\left(\frac{q_0 L}{2} + \frac{q_1 L}{6}\right) \\ 0 \\ -\left(\frac{q_0 L}{2} + \frac{q_1 L}{3}\right) \\ 0 \end{bmatrix} \tag{B.25}$$

Appendix C Fixed End Forces from Temperature Variation

C.1 Fixed End Forces from Axial Temperature Variation

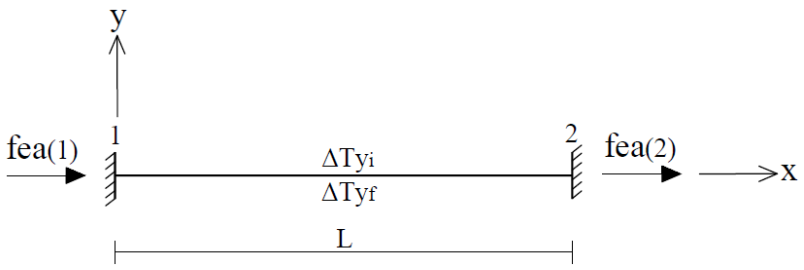

Figure C.1 Fixed end forces from axial temperature variation in the XY plane

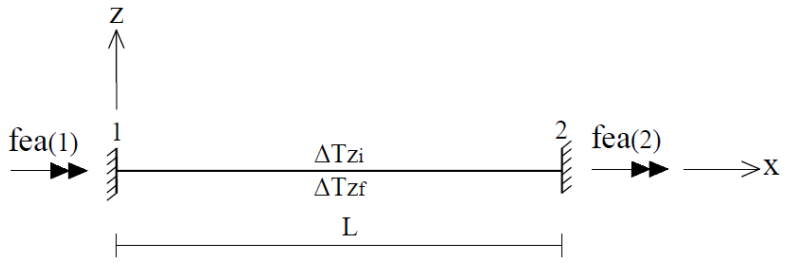

Figure C.2 Fixed end forces from axial temperature variation in the XZ plane

$$\Delta Tx_{xy} = \frac{\Delta Ty_i + \Delta Ty_f}{2} \tag{C.1}$$

$$\Delta Tx_{xz} = \frac{\Delta Tz_i + \Delta Tz_f}{2} \tag{C.2}$$

$$[fea] = \begin{bmatrix} EA \times \alpha \times \Delta Tx \\ -EA \times \alpha \times \Delta Tx \end{bmatrix} \tag{C.3}$$

C.2 Fixed End Forces from Temperature Gradient

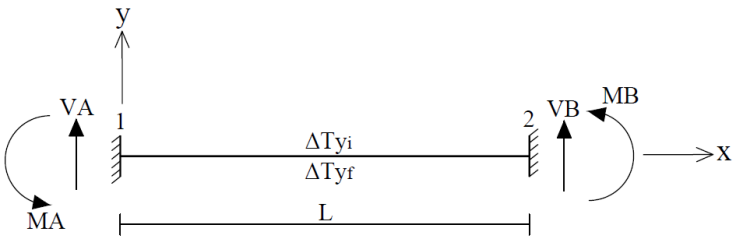

Figure C.3 Fixed end forces from temperature gradient in the XY plane

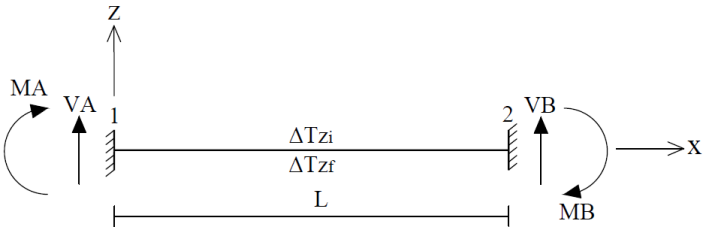

Figure C.4 Fixed end forces from temperature gradient in the XZ plane

$$\Delta Ty = \Delta Ty_f - \Delta Ty_i \tag{C.4}$$

$$\Delta Tz = \Delta Tz_f - \Delta Tz_i \tag{C.5}$$

Parameters of Timoshenko beam theory (for Euler-Bernoulli beam theory, $\Omega = 0$):

$$\Omega = \frac{EI}{G A_s L^2} \tag{C.6}$$

$$\gamma = 1 - 6\Omega \tag{C.7}$$

$$\lambda = 1 + 3\Omega \tag{C.8}$$

Fixed end forces values for an element with continuous ends:

$$\mathrm{MA_{xy}} = \frac{\mathrm{EI} \times \alpha \times \Delta \mathrm{Ty}}{h_y} \tag{C.9}$$

$$\mathrm{MA_{xz}} = -\frac{\mathrm{EI} \times \alpha \times \Delta \mathrm{T}z}{h_z} \tag{C.10}$$

$$\mathrm{MB_{xy}} = -\frac{\mathrm{EI} \times \alpha \times \Delta \mathrm{Ty}}{h_y} \tag{C.11}$$

$$\mathrm{MB_{xz}} = \frac{\mathrm{EI} \times \alpha \times \Delta \mathrm{Tz}}{h_z} \tag{C.12}$$

Transmission coefficient of moments:

$$\mathrm{tAB} = \mathrm{tBA} = \frac{\gamma}{2\lambda} \tag{C.13}$$

Element with continuous ends:

$$\left[\mathrm{fef}\right] = \begin{bmatrix} 0 \\ \mathrm{MA} \\ 0 \\ \mathrm{MB} \end{bmatrix} \tag{B.14}$$

Element with initial end hinged:

$$\left[\mathrm{fef}\right]_{\mathrm{xy}} = \begin{bmatrix} \frac{\mathrm{MA}(1 + \mathrm{tAB})}{L} \\ 0 \\ -\frac{\mathrm{MA}(1 + \mathrm{tAB})}{L} \\ \mathrm{MB} - \mathrm{MA} \times \mathrm{tAB} \end{bmatrix} \tag{C.15}$$

$$[\text{fef}]_{xz} = \begin{bmatrix} -\dfrac{MA(1+tAB)}{L} \\ 0 \\ \dfrac{MA(1+tAB)}{L} \\ MB - MA \times tAB \end{bmatrix} \quad \text{(C.16)}$$

Element with final end hinged:

$$[\text{fef}]_{xy} = \begin{bmatrix} \dfrac{MB(1+tBA)}{L} \\ MA - MB \times tBA \\ -\dfrac{MB(1+tBA)}{L} \\ 0 \end{bmatrix} \quad \text{(C.17)}$$

$$[\text{fef}]_{xz} = \begin{bmatrix} -\dfrac{MB(1+tBA)}{L} \\ MA - MB \times tBA \\ \dfrac{MB(1+tBA)}{L} \\ 0 \end{bmatrix} \quad \text{(C.18)}$$

Element with hinged ends:

$$[\text{fef}] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{(C.19)}$$

Appendix D Shape Functions

Parameters of Timoshenko beam theory (for Euler-Bernoulli beam theory, $\Omega = 0$):

$$\Omega = \frac{EI}{G A_s L^2} \tag{D.1}$$

$$\gamma = 1 - 6\Omega \tag{D.2}$$

$$\lambda = 1 + 3\Omega \tag{D.3}$$

$$\mu = 1 + 12\,\Omega \tag{D.4}$$

Element with continuous ends:

$$N1_{xy} = -\frac{x}{L} + 1 \tag{D.5}$$

$$N2_{xy} = \frac{2x^3}{L^3\mu} - \frac{3x^2}{L^2\mu} - \frac{12\,\Omega x}{L\mu} + 1 \tag{D.6}$$

$$N3_{xy} = \frac{x^3}{L^2\mu} - \frac{2\lambda x^2}{L\mu} - \frac{6\,\Omega x}{\mu} + x \tag{D.7}$$

$$N4^{xy} = \frac{x}{L} \tag{D.8}$$

$$N5_{xy} = -\frac{2x^3}{L^3\mu} + \frac{3x^2}{L^2\mu} + \frac{12\,\Omega x}{L\mu} \tag{D.9}$$

$$N6_{xy} = \frac{x^3}{L^2\mu} - \frac{\gamma x^2}{L\mu} - \frac{6\,\Omega x}{\mu} \tag{D.10}$$

$$N1_{xz} = 0 \tag{D.11}$$

$$N2_{xz} = -\frac{x^3}{L^2\mu} + \frac{2\lambda x^2}{L\mu} + \frac{6\Omega x}{\mu} - x \tag{D.12}$$

$$N3_{xz} = \frac{2x^3}{L^3\mu} - \frac{3x^2}{L^2\mu} - \frac{12\Omega x}{L\mu} + 1 \tag{D.13}$$

$$N4_{xy} = 0 \tag{D.14}$$

$$N5_{xz} = -\frac{x^3}{L^2\mu} + \frac{\gamma x^2}{L\mu} + \frac{6\Omega x}{\mu} \tag{D.15}$$

$$N6_{xz} = -\frac{2x^3}{L^3\mu} + \frac{3x^2}{L^2\mu} + \frac{12\Omega x}{L\mu} \tag{D.16}$$

Element with initial end hinged:

$$N1_{xy} = -\frac{x}{L} + 1 \tag{D.17}$$

$$N2_{xy} = \frac{x^3}{2L^3\lambda} - \frac{3x}{2L\lambda} - \frac{3\Omega x}{L\lambda} + 1 \tag{D.18}$$

$$N3_{xy} = 0 \tag{D.19}$$

$$N4_{xy} = \frac{x}{L} \tag{D.20}$$

$$N5_{xy} = -\frac{x^3}{2L^3\lambda} + \frac{3x}{2L\lambda} + \frac{3\Omega x}{L\lambda} \tag{D.21}$$

$$N6_{xy} = \frac{x^3}{2L^2\lambda} - \frac{\gamma x}{2\lambda} - \frac{3\Omega x}{\lambda} \tag{D.22}$$

$$N1_{xz} = 0 \tag{D.23}$$

$$N2_{xz} = 0 \tag{D.24}$$

$$N3_{xz} = \frac{x^3}{2L^3\lambda} - \frac{3x}{2L\lambda} - \frac{3\Omega x}{L\lambda} + 1 \tag{D.25}$$

$$N4_{xz} = 0 \tag{D.26}$$

$$N5_{xz} = -\frac{x^3}{2L^2\lambda} + \frac{\gamma x}{2\lambda} + \frac{3\Omega x}{\lambda} \tag{D.27}$$

$$N6_{xz} = -\frac{x^3}{2L^3\lambda} + \frac{3x}{2L\lambda} + \frac{3\Omega x}{L\lambda} \tag{D.28}$$

Element with final end hinged:

$$N1_{xy} = -\frac{x}{L} + 1 \tag{D.29}$$

$$N2_{xy} = \frac{x^3}{2L^3\lambda} - \frac{3x^2}{2L^2\lambda} - \frac{3\Omega x}{L\lambda} + 1 \tag{D.30}$$

$$N3_{xy} = \frac{x^3}{2L^2\lambda} - \frac{3x^2}{2L\lambda} - \frac{3\Omega x}{\lambda} + x \tag{D.31}$$

$$N4_{xy} = \frac{x}{L} \tag{D.32}$$

$$N5_{xy} = -\frac{x^3}{2L^3\lambda} + \frac{3x^2}{2L^2\lambda} + \frac{3\Omega x}{L\lambda} \tag{D.33}$$

$$N6_{xy} = 0 \tag{D.34}$$

$$N1_{xz} = 0 \tag{D.35}$$

$$N2_{xz} = -\frac{x^3}{2L^2\lambda} + \frac{3x^2}{2L\lambda} + \frac{3\Omega x}{\lambda} - x \tag{D.36}$$

$$N3_{xz} = \frac{x^3}{2L^3\lambda} - \frac{3x^2}{2L^2\lambda} - \frac{3\Omega x}{L\lambda} + 1 \tag{D.37}$$

$$N4_{xz} = 0 \tag{D.38}$$

$$N5_{xz} = 0 \tag{D.39}$$

$$N6_{xz} = -\frac{x^3}{2L^3\lambda} + \frac{3x^2}{2L^2\lambda} + \frac{3\Omega x}{L\lambda} \tag{D.40}$$

Element with hinged ends:

$$N1_{xy} = -\frac{x}{L} + 1 \tag{D.41}$$

$$N2_{xy} = -\frac{x}{L} + 1 \tag{D.42}$$

$$N3_{xy} = 0 \tag{D.43}$$

$$N4_{xy} = \frac{x}{L} \tag{D.44}$$

$$N5_{xy} = \frac{x}{L} \tag{D.45}$$

$$N6_{xy} = 0 \tag{D.46}$$

$$N1_{xz} = 0 \tag{D.47}$$

$$N2_{xz} = 0 \tag{D.48}$$

$$N3_{xz} = -\frac{x}{L} + 1 \tag{D.49}$$

$$N4_{xz} = 0 \tag{D.50}$$

$$N5_{xz} = 0 \tag{D.51}$$

$$N6_{xz} = \frac{x}{L} \tag{D.52}$$

Appendix E Internal Displacements from Linearly Distributed Load

E.1 Axial Displacement

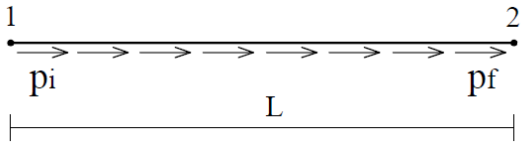

Figure E.1 Element loaded with distributed axial load

$$p_0 = p_i \tag{E.1}$$

$$p_1 = p_f - p_i \tag{E.2}$$

$$u(x) = p_0\left(-\frac{x^2}{2\,\mathrm{EA}} + \frac{L\,x}{2\,\mathrm{EA}}\right) + p_1\left(-\frac{x^3}{6\,\mathrm{EA}\,L} + \frac{L\,x}{6\,\mathrm{EA}}\right) \tag{E.3}$$

E.2 Transversal Displacement

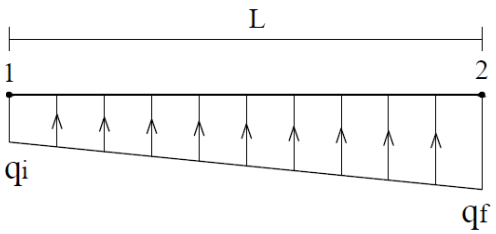

Figure E.2 Element loaded with distributed transversal load

$$q_0 = q_i \tag{E.4}$$

$$q_1 = q_f - q_i \tag{E.5}$$

$$v(x) = v_{q0}(x) + v_{q1}(x) \tag{E.6}$$

Parameters of Timoshenko beam theory (for Euler-Bernoulli beam theory, $\Omega = 0$):

$$\Omega = \frac{EI}{G A_s L^2} \tag{E.7}$$

$$\xi = 1 + 48\Omega \tag{E.8}$$

$$\zeta = 1 + \frac{40\Omega}{3} \tag{E.9}$$

$$\eta = 1 + 15\Omega \tag{E.10}$$

$$\lambda = 1 + 3\Omega \tag{E.11}$$

$$\mu = 1 + 12\Omega \tag{E.12}$$

$$\upsilon = 1 + 24\Omega \tag{E.13}$$

$$\vartheta = 1 + 8\Omega \tag{E.14}$$

$$\varsigma = 1 + 5\Omega \tag{E.15}$$

$$\psi = 1 + \frac{24\Omega}{5} \tag{E.16}$$

$$\omega = 1 + \frac{20\Omega}{9} \tag{E.17}$$

Element with continuous ends:

$$v_{q0}(x) = q_0\left(\frac{x^4}{24\,EI} - \frac{L\,\upsilon\,x^3}{12\,EI\,\mu} + \frac{L^2\,x^2\,\xi}{24\,EI\,\mu} - \frac{L^2\,\Omega\,x^2}{2\,EI} + \frac{L^3\,\Omega\,\upsilon\,x}{2\,EI\,\mu}\right) \quad \text{(E.18)}$$

$$v_{q1}(x) = q_1\left(\frac{x^5}{120\,EI\,L} - \frac{L\,x^3\,\zeta}{40\,EI\,\mu} - \frac{L\,\Omega\,x^3}{6\,EI} + \frac{L^2\,\eta\,x^2}{60\,EI\,\mu} + \frac{3\,L^3\,\Omega\,x\,\zeta}{20\,EI\,\mu}\right) \quad \text{(E.19)}$$

Element with initial end hinged:

$$v_{q0}(x) = q_0\left(\frac{x^4}{24\,EI} - \frac{L\,\vartheta\,x^3}{16\,EI\,\lambda} - \frac{L^2\,\Omega\,x^2}{2\,EI} + \frac{L^3\,x\,\xi}{48\,EI\,\lambda} + \frac{3\,L^3\,\Omega\,\vartheta\,x}{8\,EI\,\lambda}\right) \quad \text{(E.20)}$$

$$v_{q1}(x) = q_1\left(\frac{x^5}{120\,EI\,L} - \frac{L\,\varsigma\,x^3}{60\,EI\,\lambda} - \frac{L\,\Omega\,x^3}{6\,EI} + \frac{L^3\,\eta\,x}{120\,EI\,\lambda} + \frac{L^3\,\Omega\,\varsigma\,x}{10\,EI\,\lambda}\right) \quad \text{(E.21)}$$

Element with final end hinged:

$$v_{q0}(x) = q_0\left(\frac{x^4}{24\,EI} - \frac{5\,L\,\psi\,x^3}{48\,EI\,\lambda} + \frac{L^2\,\mu\,x^2}{16\,EI\,\lambda} - \frac{L^2\,\Omega\,x^2}{2\,EI} + \frac{5\,L^3\,\Omega\,\psi\,x}{8\,EI\,\lambda}\right) \quad \text{(E.22)}$$

$$v_{q1}(x) = q_1\left(\frac{x^5}{120\,EI\,L} - \frac{3\,L\,\omega\,x^3}{80\,EI\,\lambda} - \frac{L\,\Omega\,x^3}{6\,EI} + \frac{7\,L^2\,x^2}{240\,EI\,\lambda} + \frac{9\,L^3\,\Omega\,\omega\,x}{40\,EI\,\lambda}\right) \quad \text{(E.23)}$$

Element with hinged ends:

$$v_{q0}(x) = q_0\left(\frac{x^4}{24\,EI} - \frac{L\,x^3}{12\,EI} - \frac{L^2\,\Omega\,x^2}{2\,EI} + \frac{L^3\,\mu\,x}{24\,EI} + \frac{L^3\,\Omega\,x}{2\,EI}\right) \quad \text{(E.24)}$$

$$v_{q1}(x) = q_1\left(\frac{x^5}{120\,EI\,L} - \frac{L\,x^3}{36\,EI} - \frac{L\,\Omega\,x^3}{6\,EI} + \frac{7\,L^3\,x}{360\,EI} + \frac{L^3\,\Omega\,x}{6\,EI}\right) \quad \text{(E.25)}$$

Appendix F Internal Displacements from Temperature Variation

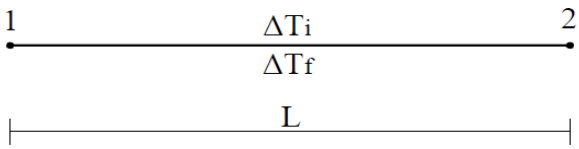

Figure F.1 Element with temperature variation

$$\Delta T = \Delta T_f - \Delta T_i \tag{F.1}$$

F.1 Axial Displacement

$$u(x) = 0 \tag{F.2}$$

F.2 Transversal Displacement

Timoshenko beam theory parameters (for Euler-Bernoulli beam theory, Ω = 0):

$$\Omega = \frac{EI}{G A_s L^2} \tag{F.3}$$

$$\gamma = 1 - 6\Omega \tag{F.4}$$

$$\lambda = 1 + 3\Omega \tag{F.5}$$

$$\mu = 1 + 12\Omega \tag{F.6}$$

Element with continuous ends:

$$v(x) = 0 \tag{F.7}$$

Element with initial end hinged:

$$v(x) = \Delta T \left(-\frac{x^3}{4 L \lambda} + \frac{x^2}{2} - \frac{L \mu x}{4 \lambda} + \frac{3 L \Omega x}{2 \lambda} \right) \tag{F.8}$$

Element with final end hinged:

$$v(x) = \Delta T \left(\frac{x^3}{4 L \lambda} - \frac{\gamma x^2}{4 \lambda} - \frac{3 L \Omega x}{2 \lambda} \right) \tag{F.9}$$

Element with hinged ends:

$$v(x) = \Delta T \left(\frac{x^2}{2} - \frac{L x}{2} \right) \tag{F.10}$$