Pedro Torres Fraga

EFFECT – Efficient Finite Element Code

Dissertação para obtenção do Grau de Mestre em
Engenharia Mecânica

Orientador: Professor Dr. João Cardoso

Júri:

Presidente:	Prof. Doutor Pedro Samuel Gonçalves Coelho
Arguente:	Prof. Doutora Marta Verdete da Silva Carvalho
Vogal:	Prof. Doutor João Burguete Botelho Cardoso

Março 2015

EFFECT – Efficient Finite Element Code

Acknowledgements

First and foremost I would like to thank Professor Dr. João Cardoso for his support and dedication, for his patience, time and knowledge that he so kindly gave to me, I am thankful. I would also like to thank him and his fellow committee members for picking me for a research grant, from which I have learned so much and was without a doubt a very excitant challenge.

I would like to thank my sister, Inês Fraga, and my father, Jorge Fraga, for their support. To my aunt, Maria de Jesus Torres, who is always ready to help me, I am thankful.

To my friends Pedro Santos, Bruno Lemos, André Águeda, João Ramalho, Marco Costa, for making the hard times easier. To João Martins who always believes in me, even when I don't. I wouldn't have made it without you. Thank you my brother.

Finally, I would like to give a very special thank you to my mother, Maria de Fatima Torres, who never lost faith in me and help me beyond her means throughout my entire academic life, thank you mother.

In loving memory of my grandmother,
Leonor da Conceição M. Torres.

Resumo

Esta dissertação tem como tema o método dos elementos finitos aplicado a estruturas mecânicas. Desenvolve-se um novo programa de elementos finitos que para além de executar diferentes tipos de análises estruturais, permite também o cálculo das derivadas de performances estruturais utilizando o método contínuo de análise de sensibilidades, com o objectivo de permitir, com o auxílio de algoritmos matemáticos existentes no programa comercial MATLAB, resolver problemas de optimização estrutural. O programa designa-se EFFECT – Efficient Finite Element Code. Recorre-se ao paradigma de programação orientada por objectos para o desenvolvimento do programa, e especificamente à linguagem de programação C++.

O objectivo principal desta dissertação é idealizar o EFFECT para que seja, nesta fase de desenvolvimento, a fundação de um programa com capacidades de análise que possam rivalizar com as de qualquer outro programa de elementos finitos de código aberto. Nesta primeira fase, são implementados 6 elementos para análise linear: elemento barra em 2 dimensões (Truss2D), elemento barra em 3 dimensões (Truss3D), elemento viga em 2 dimensões (Beam2D), elemento viga em 3 dimensões (Beam3D), elemento de casca triangular (Shell3Node) e elemento de casca quadrangular (Shell4Node). Os Elementos de casca são desenvolvidos combinando dois elementos distintos, um elemento para simular o comportamento de membrana e outro para simular o comportamento de flexão.

É também desenvolvida a capacidade de análise não linear, combinando a formulação corrotacional com o método iterativo de Newton-Raphson, mas nesta fase apenas para resolver problemas modelados com elementos Beam2D sujeitos a grandes deslocamentos e rotações, denominados problemas não lineares geométricos. A análise de sensibilidades é implementada em dois elementos, Truss2D e Beam2D, onde são incluídos os procedimentos e as expressões para o cálculo das derivadas de performances deslocamento, tensão e volume em relação a 5 diferentes tipos de variáveis de projecto. Finalmente foram criados uma série de exemplos para validar a precisão e coerência dos resultados obtidos com o EFFECT, por comparação com resultados publicados na literatura ou obtidos através do programa comercial de elementos finitos ANSYS.

Palavras-Chave: Método dos Elementos Finitos, C++, Elementos de Casca, Análise não Linear, Formulação Corrotacional.

Abstract

The theme of this dissertation is the finite element method applied to mechanical structures. A new finite element program is developed that, besides executing different types of structural analysis, also allows the calculation of the derivatives of structural performances using the continuum method of design sensitivities analysis, with the purpose of allowing, in combination with the mathematical programming algorithms found in the commercial software MATLAB, to solve structural optimization problems. The program is called EFFECT – Efficient Finite Element Code. The object-oriented programming paradigm and specifically the C ++ programming language are used for program development.

The main objective of this dissertation is to design EFFECT so that it can constitute, in this stage of development, the foundation for a program with analysis capacities similar to other open source finite element programs. In this first stage, 6 elements are implemented for linear analysis: 2-dimensional truss (Truss2D), 3-dimensional truss (Truss3D), 2-dimensional beam (Beam2D), 3-dimensional beam (Beam3D), triangular shell element (Shell3Node) and quadrilateral shell element (Shell4Node). The shell elements combine two distinct elements, one for simulating the membrane behavior and the other to simulate the plate bending behavior.

The non-linear analysis capability is also developed, combining the corotational formulation with the Newton-Raphson iterative method, but at this stage is only avaiable to solve problems modeled with Beam2D elements subject to large displacements and rotations, called nonlinear geometric problems. The design sensitivity analysis capability is implemented in two elements, Truss2D and Beam2D, where are included the procedures and the analytic expressions for calculating derivatives of displacements, stress and volume performances with respect to 5 different design variables types. Finally, a set of test examples were created to validate the accuracy and consistency of the result obtained from EFFECT, by comparing them with results published in the literature or obtained with the ANSYS commercial finite element code.

Keywords: Finite Element Method, C++, Shell Elements, Nonlinear Analysis, Corotational Formulation.

Table of Contents

List of Figures

List of Tables

List of Symbols

x, y, z	Cartesian coordinates
ξ, η, ζ	Natural coordinates of isoparametric elements
u, v, w	Displacement components in coordinate directions
$\theta_x, \theta_y, \theta_z$	Rotation components about coordinate axes
$\lambda_x, \lambda_y, \lambda_z$	Direction cosines of the local coordinate axes
V_x, V_y, V_z	Vectors used to define the local directions
A	Area of the cross section
$a(z, \bar{z})$	Term associated with the elastic energy
b	Design variables vector
b_i^-, b_i^+	Bounds of the design variable
E	Modulus of elasticity of the material
$F(b)$	Objective function
G	Modulus of rigidity of the material
J	Determinant of the Jacobian
κ	Curvature
L	Length of the element
M	Bending moment
N	Normal force
Q_k	Equality constraint
P	External load
P_j	Inequality constraint
t	Thickness; time
$T(x, \tau)$	Transformation associated with the shape change
U	Bending energy
$v(x)$	Lateral displacement function
V	Velocity field
V_r	Auxiliary vector
V_s	Shear force
W	Energy due an external load
x	Position vector
x_t	Position vector on the transformed domain
ż	Material derivative
$\bar{z}$	virtual displacement vector
{a}	Generalized degrees of freedom
$\{e_{Pb}\}$	Imbalance loads vector
$\{R\}$	External load vector
$\{U\}, \{u\}, \{u^{tot}\}$	Displacement vector
$\{U_b\}$	Bending degrees of freedom
$\{U_m\}$	Membrane degrees of freedom

$\{u^{def}\}$	Displacements causing deformation
$\{u^{rig}\}$	Displacements associated with the rigid body motion
$\{X_g\}$	Element coordinates on the global system
$\{\mathrm{X'_g}\}$	Undeformed position expressed in the rotated corotational frame
$\{X_e\}$	Element coordinates on the local system
[B]	Stain-displacement matrix
$[D_b]$	Matrix of elestic stiffnesses for the DKT and QKT element
$[E]$	Matrix of elestic stiffnesses
$[G]$	Geometric stiffness matrix
$[N]$	Shape functions
$[H_x], [H_y]$	Component vectors of the shape functions
$[k]$	Matrix of curvatures
$[K_b]$	Stiffness matrix of a plate bending element
$[K_g]$	Element stiffness matrix in the global coordinate system
$[K_L], [K]$	Element stiffness matrix in the local coordinate system
$[K_m]$	Stiffness matrix of a membrane element
$[K_{qfs}]$	Stiffness matrix at each node of a flat shell element
$[K_t]$	Tangent stiffness matrix
$[T]$	Transformation matrix
$[\lambda]$	Transformation matrix
$\{\varepsilon\}$	Strains vector
$\{\Delta P\}$	Load increment vector
$\{\Delta U\}$	Displacement increment
$\{\theta^{def}\}$	Rotation component causing deformation
$\{\theta^{rig}\}$	Rotation associated with the rigid body motion
$\{\theta^{tot}\}$	Total rotations vector
β_x	Rotations in the direction normal to the x-z planes
β_y	Rotations in the direction normal to the y-z planes
ε, γ	Strains
λ	Adjoint displacements
$\delta(x-\hat{x})$	Dirac function
τ	Transformation parameter
σ	Stress matrix
υ	Poisson's coefficient
$\Psi(b)$	Performance
Ψ_b	Gradient of Ψ regarding b
Ω	Domain of the problem
Ω_τ	Domain transformed by $T(x,\tau)$

Chapter 1

1. Introduction

1.1. Overview

The Finite Element Method (FEM) is a very effective numerical procedure used to solve real-world engineering problems. In the finite element method a continuum domain of a problem is subdivided into finite portions, changing its nature to discrete and thus making it much easier to solve. Normally, the larger the number of portions, the better is the approximation to reality. The portions are called finite elements and when using FEM in solid mechanics a structure is always divided into a finite number of elements delimited by nodes.

FEM is a very powerful method, as it provides the capacity of solving a great number of engineering problems such as structural analysis (determination of stresses, strains and natural frequencies), heat transfer and fluid flow. Coupled with the advancement of digital computers in the last decades, it became a generic and very popular tool among engineers, since it allows creating numerical models of real structures or components easily and then obtaining estimates of their behavior.

In structural mechanics, models can be linear or nonlinear, depending on the assumptions made. On a great number of problems, linear analysis gives results sufficiently approximate to the reality. However, it is inevitable in some scenarios the transition to nonlinearity. For example, in the field of structural mechanics, the stiffness and even loads may become dependent on the displacements or the deformations. In those situations it is impossible to solve directly the equations of equilibrium that describe the structural behavior, since they have to be written in terms of the deformed configuration, so it becomes necessary to employ an iterative method. These kind of methods need to know and update the equilibrium positions in each iteration, rewriting the structural equations. The iterative process stops when a convergence criterion is met and the structure arrives at its final deformed state for the prescribed applied load.

To solve nonlinear problems of geometric nature, i.e., when equilibrium equations must be written in the deformed configuration, it becomes necessary to employ a kinematic description together with an iterative methods. There are mainly three types of these kinematic formulations: (1) total Lagrangian, (2) updated Lagragian and (3) corotational formulation. They are distinguished basically by the reference frame. The corotational formulation, kinematic description applied in this thesis, is the most recent of the three and has the particularity of using three coordinate systems or frames: A global frame, used to measure the displacements of the nodes, a local frame and the corotated frame, that is obtained from the local frame through a rigid body motion and is used to measure stresses and deformations.

Structural optimization has been the focus of an extensive research effort since it was first used 40 years ago, not only because of the clear advantages it represents in terms of cost reduction in industrial projects, especially in automotive, aeronautical and aerospace industry, but also because it also can contribute to substantially speed-up the design process, allowing to quickly get the best settings for the design variables, i.e., the ones that correspond to the most

appropriate solutions. Structural optimization combines optimization algorithms with FEM numerical methods and a very useful and practical feature that can be implemented into a finite element program, taking advantage of its structural analysis capacity, is an interface that allows it to be used with an optimization algorithm to solve structural optimization problems.

For structural optimization it is meant the ability to perform a set of operations to automatically find the optimal values of the parameters that improve a given mechanical structure, according to a prescribed objective and satisfying pre-established constraints. Design sensitivity analysis can compute gradients of the functions used in the formulation of structural optimization problems with respect to the design variables and these gradients can be employed by optimization algorithms to obtain the solution. As the sensitivity analysis can significantly improve the efficiency of optimization algorithms, it is studied in this dissertation.

1.2. Objectives

The main goal of this dissertation is the development of a C++ program capable of executing structural analysis based on the finite element method and capable of performing sensitivity analysis for optimization purposes. It was decided to name this software EFFECT – Efficient Finite Element Code.

The first objective defined was the search for FEM open source codes available for download and study them. The FEM has been the focus of intense research effort in the past and several researchers have contributed to create and improve FEM open codes and publish about them, and the intention here was to take advantage of the work that has been developed and learn as much as possible from it, allowing to better understand how a FEM program should be structured and constructed. From the FEM codes found available for download the OOFEM - Object Oriented Finite Element Solver [1] was considered the most complete.

It was decided that EFFECT should have 6 different finite element types in this first version: two-dimensional truss (Truss2D), three-dimensional truss (Truss3D), two-dimensional beam (Beam2D), three-dimensional beam (Beam3D), triangular flat shell (Shell3Node) and quadrilateral flat shell element (Shell4Node). These elements and the corresponding Degrees Of Freedom (DOF) are shown in table 1.1.

Table 1.1 – EFFECT's elements and associated DOF.

Element	DOF on the local frame	DOF on the global frame
Truss2D	2	4
Truss3D	2	6
Beam2D	6	6
Beam3D	12	12
Shell3Node	18	18
Shell4Node	24	24

Of the six element types, two present a substantially higher level of complexity on their formulation, the triangular and quadrilateral flat shell elements. The reason for this complexity is that both elements are in fact an assembly of more than one element.

The triangular flat shell element is built by combining the Constant Stain Triangle (CST) element to simulate the membrane behavior and the Discrete Kirchhoff Theory (DKT) element developed by Batoz, Bathe and Ho [2] for the plate bending behavior. The quadrilateral flat shell is also composed from two different elements, the membrane behavior is modeled by the isoparametric four node quadrilateral element (Q4) and the plate bending behavior is modeled by the Discrete Kirchoff Quadrilateral (DKQ) element developed by Batoz and Tahar [3].

With the 6 element types mentioned it is possible to model a structure with linear behavior. Nonlinear analysis capabilities are considered, but only for structures modeled with the Beam2D element. With EFFECT it is possible to solve 2D problems modeled with beam elements subjected to large displacements and rotations. To solve this kind of problems the corotational formulation and the Newton-Raphson method were implemented.

EFFECT can be used as part of an integrated system designed for structural optimization, together with a structural reliability analysis program and the commercial software MATLAB. To accomplish this goal, the capability of performing design sensitivity calculations is implemented in EFFECT for the Truss2D and Beam2D elements regarding 5 different design variables types. For this integration to be operational, an interface is also developed to allow exchanging information with the reliability algorithm and the algorithms needed to perform optimization, that are included in the MATLAB Toolbox *optimtool*.

The global objective that lead to the creation of this new software based on the finite element method was to provide a basis to be used and further extended by students and researchers in the areas of FEM, structural optimization, sensitivity analysis or reliability analysis. The focus was put in assembling together all the functionalities mentioned before instead of developing each one individually.

1.3. Organization

This dissertation is divided in ten chapters, including the Introduction. In chapter 2 an overview of the subjects addressed in this dissertation will be given, focusing on the evolution of the triangular and quadrilateral flat shell elements, the corotational formulation and the C++ implementation of FEM.

Chapters 3 to 5 present the formulation of the elements implemented in EFFECT. Chapter 3 describes Truss2D, Beam2D, Truss3D and Beam3D elements. These four different types of elements are set together in the same chapter because they actually are not so different from each other; they even share portions of their individual formulations among themselves. For instance, part of the stiffness matrix of the Beam2D element, is composed by the Truss2D matrix formulation.

Chapter 4 presents the triangular flat shell element implemented. As mentioned before, the triangular flat shell element is composed from two elements: the CST and the DKT element, their formulation is completely developed in this chapter as well as the explanation of how these

two elements link together and their combined behavior. Chapter 5 presents the formulation for the quadrilateral flat shell and the elements that compose it: the isoparametric four node quadrilateral (Q4) element and the DKQ element,

In chapter 6 the nonlinear analysis is discussed. The chapter begins with an explanation about the concept of nonlinearity and the types of problems it is intended to deal with , then the Newton-Raphson iterative method is studied followed by the development of the corotational formulation. Chapter 7 demonstrates how EFFECT can be used in the context of structural optimization. It is shown how to formulate an optimization problem with the help of MATLAB, and the calculations needed to obtain the performance gradients regarding the design variables.

Chapter 8 is of the most importance, because it is where the C++ code of EFFECT is fully explained. It is intended to completely dissect the software to its most basic object and show how they interact with each other. EFFECT's running behavior will also been explained and its capability will be fully described. Chapter 9 presents several test cases used to verify the accuracy of the results and validate the program. Finally a brief summary of all the work done to create EFFECT is given in the chapter 10.

Chapter 2

2. Literature Review

2.1. Overview

This chapter presents concepts and briefly reviews the literature on several domains that are essential for the developement of this thesis. It addresses shell elements, the corotational formulation for structural nonlinear analysis, object-oriented programming applied to the Finite Element Method (FEM) structural optimization and design sensitivity analysis

These topics where chosen due to their importance. However, programming the FEM involves a variety of other subjects, which are not covered in this chapter because they correspond to consolidated scientific knowledge, or simply because it seems reasonable to limit the thesis size. Yet, the FEM deserves a brief review.

The finite element method has its origins in the early 1950s, but its formal presentation as we know it today with the direct method for assembly elements stiffness matrices is attributed to Turner, Clough, Martin and Toper [4] and apparently the term "finite element" was introduced by Clough [5]. Alongside with the mentioned authors, several other important researchers also had an enormous contribution on the appearance, development and divulgation of the FEM with their papers and publications, including Courant [6], Argyris [7] and Zienkiewicz and Cheung [8].

2.2. Shell Elements Literature Review

According to Cook [9], currently there are 3 types of approaches to the development of shell elements:

(1) Flat shells, formed by combining a plane membrane elements with a plate bending elements (low-order isoparametric elements);
(2) Degenerating 3D solid elements (high-order isoparametric elements);
(3) Curved elements based on classical shell theory.

From the three approaches the third one has been receiving less attention nowadays, although it has completely dominated, together with the second approach, for over one decade, during the 1970s. The first approach only recently started to get back into the spotlight thanks to new and more effective plate bending elements. According to Bathe and Ho [10] the second approach produces very effective elements but very demanding computationally, due to the large size of the stiffness matrices. On the other hand, the elements from the first approach present good results and are rather inexpensively when compared with the higher-order isoparametric elements.

One of the biggest problems when assembling the stiffness matrices of membrane and plate bending elements, according to Zienkiewicz and Taylor [11], is the creation of a zero on the drilling degree of freedom, which causes a singularity in the global stiffness matrix when the

elements are coplanar and there is no coupling between the membrane and bending stiffness of the element. They present two simple methods to fix the problem:

(1) Assemble the equations at points where elements are co-planar in local coordinates and delete the equation regarding the shell-normal rotation ($\theta_z = 0$);
(2) Introduce an arbitrary small value to the drilling degree of freedom.

However Zienkiewicz and Taylor [11] make note that those two solutions present programing difficulties due to the fact that is necessary to assess if the elements are co-planar or not. They proposed to add to the formulation of each element the term,

$$\Pi^* = \Pi + \int_{\Omega} \alpha_n E\, t^n (\theta_z - \overline{\theta_z})^2 d\Omega \tag{2.1}$$

where α_n is a ficticius elastic parameter, $\overline{\theta_z}$ is a mean rotation of each element which permits the element to satisfy local equilibrium in a weak sense and t^n is a scalling value, in order to avoid the dependency on the geometry of the elements.

In this thesis two flat shells elements are implemented, not only because they are the easier ones to implement but also because they give good results in comparison to the high-order isoparametric elements.

2.2.1. Triangular Flat Shell Elements Literature Review

The concept of triangular flat shells can be traced back to 1961, been introduced by Greene [12]. However the results obtained by those elements weren't at all satisfactory, due in large part to lack of development of the stiffness matrices for the plate bending portion available at that time. One of the first triangular plate bending elements created, was developed by Clough and Tocher [13] in 1965, by dividing each triangle about its centroid into three sub triangles.

Later on, Batoz *et al.* [2] developed three types of plate bending elements:

(1) the DKT element based on Discrete Kirchoff Theory assumptions;
(2) the HSM element based on the Hybrid Stress Method;
(3) the SRI element based on Selective Reduced Integration scheme.

After Batoz *et al.* [2] compared the results obtained for these elements; they concluded that the DKT and the HSM elements presented better results and show better reliability.

Bathe and Ho [10] developed a flat shell triangular element similar to the one implemented on this thesis, combining the CST element for membrane stiffness with a plate bending element based on Mindlin theory of plates, similar to the DKT element from Batoz *et al.* [2], for the bending stiffness. For the drilling degrees of freedom they introduced a fictitious stiffness. Bathe and Ho [10] concluded that the element presented excellent bending properties, the shortcoming was the CST element, due to the membrane stresses been assumed as constant, but in overall they found the element to be very reliable.

2.2.2. Quadrilateral Flat Shell Element Literature Review

The membrane part of the quadrilateral flat shell element implemented in EFFECT is the isoparametric quadrilateral (Q4) element. The concept of isoparametric elements was introduced by Irons [15], being the Q4 element one of simplest of those elements. The shape functions used to formulate the element were developed by Ergatoudis, Irons and Zeinkiewicz [16], according to Kamara [17].

One of the first isoparametric quadrilateral shell elements was developed by McNeal [18] in 1978, called QUAD4. The QUAD4 element is a four-node thick-shell isoparametric element, since it was formulated based on isoparametric shape functions, but the excessive constraints were relaxed. As McNeal [18] notices, the standard thick-shell isoparametric element, up to that date did not show particularly good accuracy. However the QUAD4 element presented a consistent formulation of membrane and bending strains with a reduced order integration scheme for shear terms, which increased substantially the accuracy of the element for the plate bending action.

Although good membrane elements emerged soon after the introduction of the finite element method, the plate bending elements took a little longer, as problems like shear locking were compromising the effectiveness of such elements. A major development was the Mixed Interpolated Tensorial Components (MITC) elements developed by Bathe and Dvorkin [19], the shear locking problem has solved by including the shear and bending effects through different interpolations. One specific element of the MITC family worth mentioning is the MITC4 plate bending element, that even though having a configuration identical or near to other elements, like for instance the QUAD4 element of McNeal [18], can also be used for nonlinear analysis while other elements cannot. The MITC4 element showed optimal convergence behavior and accuracy.

The plate bending element used for the quadrilateral flat shell element in this thesis was developed by Batoz and Tahar [3]. Like the DKT element, the QKT formulation was based on the discrete Kirchoff theory and transverse shear strain was also neglected.

2.3. Corotational Formulation Literature Review

The Corotational formulation has its origins on a concept with more than two centuries old, the polar decomposition theorem. With this theorem it is possible to decompose the total deformation of a continuous body into a rigid body motion and a purely deformational component; In the 1950s and 1960s this "rigid-plus-deformational" approach began to be used in the aerospace industry, where only one cartesian system is defined for the entire structure, which follows the body while it deforms. This cartesian system is called corotated configuration or in some literature the shadow frame.

In the early 1960s Argyris *et al.* [21] presented a method of calculating the geometrical stiffness matrices by using the rigid-plus-deformational approach, which he first called the “natural approach”. Although apparently the first FEM paper with “corotational” in its title was written by Belytschko and Glaum [22], the corotational formulation was first introduced into the finite element analysis by Wempner [23] and Belytschko and Hsieh [24]. While Wempner [23] developed the formulation to deal with shell elements under small deformation and large

displacements, Belytschko and Hsieh [24] worked on the formulation for beam elements under large rotations and developed a convected coordinate system.

The realm of problems in which the corotational formulation was used, but only a single set of cartesian axis is defined, was well described by Fraeijs de Veubeke [25]. In those problems the intention was finding the mean motion of the body so that the corotated or shadow configuration could be located and oriented. However the use of one single frame for the entire structure was not practical for FEM analysis, especially to fulfill the assumption in which the corotation formulation is based, that although displacements can be large the deformations must remain small. Horrigmoe and Bergan [26] introduced the notion that instead of having just one frame for the entire structure, an individual corotated frame could be attached to each element.

In 1986 Rankin and Brogan [27] introduced the concept of element independent corotational formulation (EICR), in which are not exactly used corotated configurations, but a kind of projections. Rankin and Brogan [27] noticed that the corotational capabilities were until that date an integral part of the element formulation and presented a new implementation procedure for development of the corotational formulation which is independent of the element making it possible to give to an element the corotational capabilities without having to update its shape functions. In this thesis the corotational formulation for the deformed displacements is based on the work of Rankin and Brogan [27], since it allow the reuse of the linear analysis capabilities pre-implemented of the elements.

In 2005, Felippa and Haugen [28] presented a unified theoretical framework for the corotational kinematics formulation directed for geometrically nonlinear analysis, assuming small strains and elastic material behavior. They also proposed future improvements to the theory including the relaxing of the previous small strain assumption to allow moderate deformations, or improvement of the handling of extremely large rotations involving multiple revolutions.

2.4. C++ Programming Language Literature Review

Actually there are 2 types of approaches regarding the programming paradigm applied in FEM program design: (1) procedural programming and (2) object-oriented programming. The procedural approach until very recently had very strong support, since it already had proven its effectiveness in dealing with complex numerical operations. Although languages like FORTRAN, Pascal or C, which fall under this category of procedural paradigm, were very appreciated for their speed and capabilities as reliable numerical tools they weren't suitable for writing increasingly large and complex programs and here is where the object-oriented programming paradigm takes over.

The object-oriented programming paradigm, associated for instance with the C++ and Java programming languages, has proven itself in recently years to be quite suitable to create large and robust scientific computers software, allowing a team of developers to work simultaneously in the same software. There are essentially 3 key features which are greatly responsible for the success of this paradigm:

(1) Encapsulation – this feature refers to the possibility of grouping together both data and procedural tasks in one single object. Objects are self-contained entities, capable to

store data through their variables and executing tasks through their methods. Similar Objects either in terms of behavior or functionality can be also grouped together in classes. An object can at any time access other Object by sending it a message, making the receiver execute the requested method.

(2) Inheritance – this feature refers to the capacity that the object-oriented languages have to define subclasses that automatically have direct access (inherit) to the variables or methods of the base class. The subclasses can also share the same method but it can execute different tasks in each one of them. The possibility to define a hierarchy of classes allows optimal organization and reuse of the code.

(3) Polymorphism – this feature refers to the ability of two Objects to respond to the same message but reacting in their own manner.

According to Yves Dubois-Pèlerin and Pierre Pegon [29], the publications in which the key concepts of the object-oriented paradigm (encapsulation, inheritance, and polymorphism) were first applied to the finite element analysis can be traced back to books like those of Meyer [30] and Cox [31] and papers like those of Fenves [32] and Miller [33]. Fenves [32] point out the clear advantages of object-oriented programming and Miller [33] first presented the degree-of-freedom, the node and the element as the basic objects of an object-oriented finite element program.

In 1990 Forde, Foschi, and Stiemer [34] released one of the first detailed implementations using the object-oriented paradigm for FEM applications. Forde *et al.* [34], worked on two-dimensional problems where the need for new basic objects emerged, and created a structure consisting of: elements, nodes, materials, boundary conditions and loads components. At that time many authors began to study the subject and presented different architectures of object-oriented finite element codes, two great developments where from Zimmermann, Dubois-Pelerin and Bomme [35] and Miller [36].

Zimmermann *et al.* [35] developed the structure for a linear dynamic finite element analysis program. The structure was divided in three categories, where they separated the FEM Objects (or discretization of the model) from the algebraic features. By defining this structure they developed the concept of the "non-anticipation" principle. They first used the object-oriented language Smalltalk to implement the program and only later on the C++ environment. Miller [36] developed the structure of his nonlinear dynamic analysis program where he designs a coordinate free approach.

Mackie [37] first releases an introduction to the object-oriented programming regarding finite element analysis, where he illustrates the importance of the ability to define a hierarchy of classes and subsequently of the inheritance feature of this paradigm. Mackie [38] later on also published his own system data structure, which according to him, enables to take advantage of the object-oriented paradigm to better handle with the ever growing complexity of the finite element analysis software. Archer, Fenves and Thewalt [39] review some of the structures already available at that time and presented a new architecture for finite element software to include nonlinear analysis of structures under static and dynamic loads.

At last, Patzák and Bittnar [40] publish their own structure of the "Object Oriented Finite Element Modeling" (OOFEM) software. OOFEM was a source of great inspiration for the creation of EFFECT, its structure was largely based on the one found in OOFEM.

2.5. Optimization and Sensitivity Analysis Literature Review

The first developments in structural configuration optimization using mathematical programming were accomplished by Dorn, Gomory and Greenberg [41]. However they focus primarily on statically determinate structures composed by truss elements and only subject to a single load. Dobbs and Felton [42] took a step forward by including ground structures statically indeterminate and subject to multiple loads conditions. Vanderplaats and Moses [43] presented a method to obtain the minimum weight of a structure regarding geometry variables. In 1986, Haug, Choi and Komkov [44] released a methodology in which the concept of material derivate and the adjoint method are used together to obtain the analytic sensitivity expressions regarding shape variations.

Even though the optimization techniques based on the continuum method were already in a good stage of development, they were greatly directed to the truss structures or ground structures. In 1992, Twu and Choi [45] based on the work of Haug *et al.* [44] published a sensitivity analysis method for a more general type of structures that include truss, beam, plane elastic solid and plate components. In their method they studied separately the shape and the orientation variations of the component. To account for the effects of shape variation they used the material derivative approach and for the effects of the orientation variation they used a similar procedure to the continuum shape design sensitivity analysis method.

Cardoso [46] presented a methodology to obtain the analytic expressions for sensitivity calculation, based on the work of Haug *et al.* [44] allowing the application the continuum method to handle the variations in shape and rotations. Cardoso [46] notes that even though the expressions obtained are simpler than the ones from Twu and Choi [45] all the non-null terms obtained from them are included in his expressions. In this thesis the expressions from Cardoso [46] are implemented in EFFECT.

Chapter 3

3. Truss and Beam Elements Formulation

3.1. Overview

This chapter presents the formulation of the stiffness matrix of the truss and beam elements. These are the simplest elements to implement in a computer program based on the finite element method. In particular the Truss2D element was selected as the first to be implemented, and its development actually serves a double purpose, it allows analyzing truss structures and also, by being the first element to be implemented, enables to check and evaluate if all of the program functionalities are working correctly.

The truss elements only accounts for the internal energy associated with axial forces acting on the element. For this reason they only have one degree of freedom in each node on their local coordinate system, the axial displacement. The difference between the Truss2D and Truss3D elements is just the size of the stiffness matrix and the coordinate transformation system, the local stiffness matrix formulation remains the same.

The beam elements on the other hand account for the internal energy associated to axial forces and bending moments. Consequently, the Beam2D element has 3 degrees of freedom on its local coordinate system, for each node. The Beam3D, besides, considers internal energy associated with torsion and has 6 degrees of freedom per node.

3.2. Truss2D and Truss3D Stiffness Matrix Formulation

The Truss2D element formulation was obtained by using the isoparametric formulation. An isoparametric element is an element which uses the same shape functions to interpolate both the nodal coordinates and the displacements and its geometry has to be defined by a natural coordinate system, from which the shape function matrices are function of. The Truss2D and its natural coordinate system are shown in the figure 3.1.

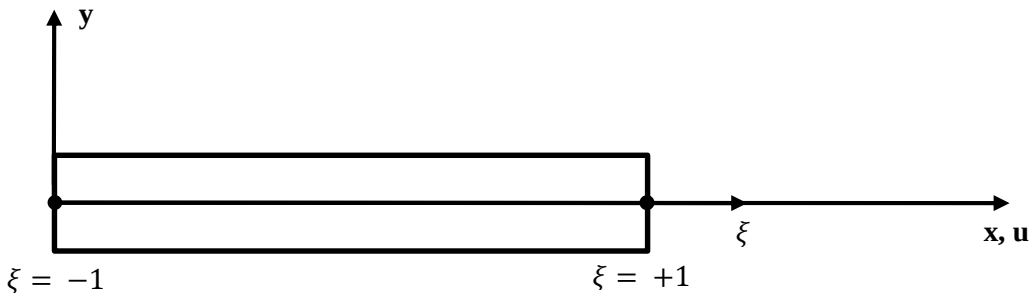

Figure 3.1 - Truss2D element on the global and natural coordinate system.

The Truss2D element only has two degree of freedom, the axial displacement at each node,

$$\{U\}^T = \{u_1 \quad u_2\}$$

Due to this fact, it is only necessary one natural coordinate to describe the element (ξ), which is associated with a single axis, that runs through the length of the element and remains independent of the element orientation on the global coordinate system, as shown in figure 3.1. It is important to notice that this coordinate varies between $\xi = -1$ on node 1 and $\xi = +1$ on node 2, and these are the boundary conditions necessary to obtain the shape functions. The formulation of this element stiffness matrix has been developed by considering the infinitesimal strain-displacement relationships, as it follows.

The relationship between the global and the natural coordinates system can be obtained by interpolation, as follows:

$$x = a_1 + \xi a_2 \qquad or, \qquad x = [1 \quad \xi] \begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix} \tag{3.1}$$

the displacements are obtained similarly,

$$u = a_1 + \xi a_2 \qquad or, \qquad u = [1 \quad \xi] \begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix} \tag{3.2}$$

applying boundary conditions,

$$\xi = -1 \Rightarrow x_1 = a_1 - a_2$$

$$\xi = +1 \Rightarrow x_2 = a_1 + a_2$$

or in matrix form,

$$\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix} \qquad or, \qquad \{x\} = [A] \begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix} \tag{3.3}$$

by inverting $[A]$ and using the equation (3.1) one obtains (for the displacement the procedure is analogous),

$$x(\xi) = [1 \quad \xi] \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} \qquad or, \quad x(\xi) = [N] \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} \tag{3.4}$$

where $[N]$ are the shape functions,

$$[N] = [1 \quad \xi] \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{3.5}$$

or,

$$N_1 = \frac{1}{2}(1 - \xi) \tag{3.6}$$

$$N_2 = \frac{1}{2}(1 + \xi)$$

The axial strain in the element is,

$$\varepsilon_x = \frac{\partial u}{\partial x} \tag{3.7}$$

However, in an isoparametric element the geometry is defined in the natural coordinate system, the du/dx derivative is not available directly, it is then necessary to apply the chain rule,

$$\frac{\partial u}{\partial \xi} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial \xi} \tag{3.8}$$

or,

$$\frac{\partial u}{\partial \xi} = J\frac{\partial u}{\partial x} \tag{3.9}$$

where J is the Jacobian,

$$J = \frac{dx}{d\xi} = \frac{d}{d\xi}[N]\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \end{bmatrix}\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \frac{x_2 - x_1}{2} = \frac{L}{2} \tag{3.10}$$

where the L is the length of the element. Inverting equation (3.9),

$$\frac{\partial u}{\partial x} = \frac{1}{J}\frac{\partial u}{\partial \xi} \tag{3.11}$$

by combining (3.7) and (3.11) one obtains,

$$\{\varepsilon\} = \frac{\partial u}{\partial x} = \frac{1}{J}\frac{\partial u}{\partial \xi} = [B]\{u\} \tag{3.12}$$

where $[B]$ is the stain-displacement matrix,

$$[B] = \frac{1}{J}\frac{d}{d\xi}[N] = \frac{1}{J}\begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{3.13}$$

using equation (3.10),

$$[B] = \frac{1}{L}\begin{bmatrix} -1 & 1 \end{bmatrix} \tag{3.14}$$

the truss stiffness matrix is then as follows,

$$[K] = \int_0^L [B]^T[B]EA\, dx = \int_{-1}^{1} [B]^T[B]EA\, J d\xi \tag{3.15}$$

by using the equation (3.14) the equation (3.15) becomes,

$$[K] = EA\int_{-1}^{1} \begin{bmatrix} -1 \\ 1 \end{bmatrix}\begin{bmatrix} -1 & 1 \end{bmatrix} J\, d\xi \tag{3.16}$$

where E is the Young's modulus of the material and A the area of the cross section of the element. Finally using one point gauss quadrature, the stiffness matrix is obtained for the Truss2D,

$$[K] = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{3.17}$$

The Truss3D the stiffness matrix is the same as the one shown in (3.17). The differences only appear when the coordinate transformation is applied, and that will be explained later.

3.3. Beam2D and Beam3D Stiffness Matrix Formulation

The Beam2D element accounts for the internal energy associated to axial forces and bending moments. Assuming the two terms are mutually independent, two different formulations can be derived independently. This is aided because the element has 3 degrees of freedom per node, axial displacement, u, transversal displacement, v, and cross-section rotation, θ_z, and the fact that the energy associated to axial forces is only associated with u and the energy associated to bending is exclusively related with v and θ_z. The assembly produces an element with 6 degrees of freedom:

$$\{U\}^T = \{u_1 \quad v_1 \quad \theta_{z1} \quad u_2 \quad v_2 \quad \theta_{z2}\}$$

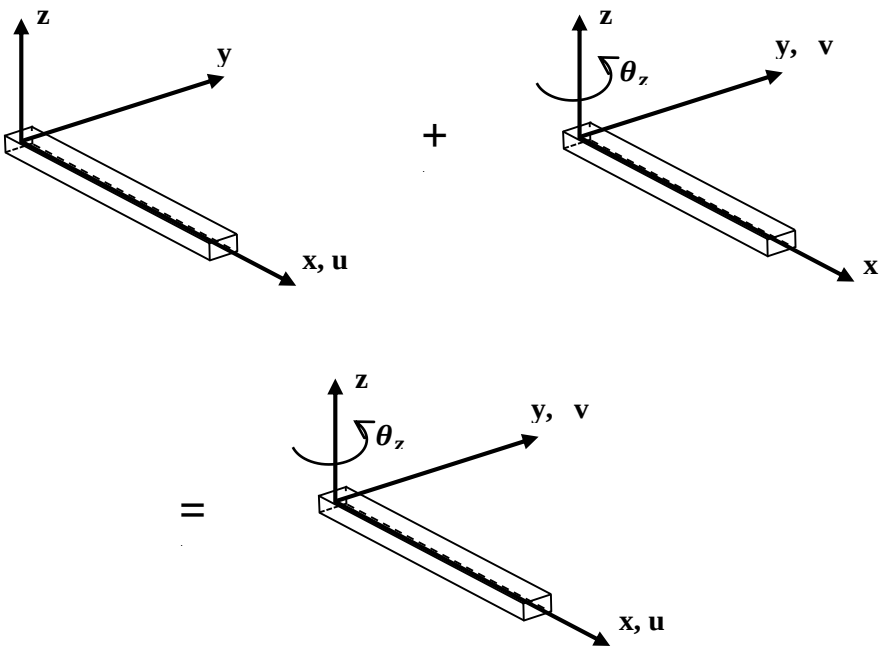

Figure 3.2 – Combination of 2 different DOF sets to obtain the Beam2D element.

Since the axial forces have already been accounted for in the formulation of the truss elements, only the part of the stiffness matrix that accounts for the internal energy due to bending will be derived. A different approach will be used, considering the bending moment *M* and the curvature *κ*, instead of the relationships between the displacement and strain like in Truss2D. Also, the Euler-Bernoulli theory that does not account for shear strain is used here.

$$M = EI_z\kappa \qquad \text{where, } \kappa = \frac{d^2v}{dx^2} \tag{3.18}$$

where *v= v(x)* is the lateral displacement,

$$v(x) = a_1 + a_2x + a_3x^2 + a_4x^3 \qquad or, \qquad v(x) = [1 \quad x \quad x^2 \quad x^3]\begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{Bmatrix} \tag{3.19}$$

applying the boundary conditions,

$$x = 0 \Rightarrow v_1 = a_1$$

$$x = 0 \Rightarrow \frac{dv}{dx} = \theta_{z1} = a_2$$

$$x = L \Rightarrow v_2 = a_1 + a_2L + a_3L^2 + a_4L^3$$

$$x = L \Rightarrow \frac{dv}{dx} = \theta_{z2} = a_2 + 2a_3L + 3a_4L^2$$

or in matrix form,

$$\begin{Bmatrix} v_1 \\ \theta_{z1} \\ v_2 \\ \theta_{z2} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & L & L^2 & L^3 \\ 0 & 1 & 2L & 3L \end{bmatrix}\begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{Bmatrix} \qquad or, \qquad \{u\} = [A]\begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{Bmatrix} \tag{3.20}$$

by inverting $[A]$ and using equation (3.19) one obtains,

$$v = [1 \quad x \quad x^2 \quad x^3]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{L^2} & -\frac{2}{L} & \frac{3}{L^2} & -\frac{1}{L} \\ \frac{2}{L^3} & \frac{1}{L^2} & -\frac{2}{L^3} & \frac{1}{L^2} \end{bmatrix}\begin{Bmatrix} v_1 \\ \theta_{z1} \\ v_2 \\ \theta_{z2} \end{Bmatrix} \qquad or, \qquad v = [N]\begin{Bmatrix} v_1 \\ \theta_{z1} \\ v_2 \\ \theta_{z2} \end{Bmatrix} \tag{3.21}$$

where $[N]$ are the shape functions,

$$N_1 = 1 - \frac{3}{L^2}x^2 + \frac{2}{L^3}x^3$$

$$N_2 = x - \frac{2}{L}x^2 + \frac{1}{L^2}x^3 \tag{3.22}$$

$$N_3 = \frac{3}{L^2}x^2 - \frac{2}{L^3}x^3$$

$$N_4 = -\frac{1}{L}x^2 + \frac{1}{L^2}x^3$$

since the curvature is also given by,

$$\kappa = [B]\{u\} \tag{3.23}$$

combining with (3.18) is obtained,

$$\frac{d^2v}{dx^2} = [B]\{u\} \Leftrightarrow \frac{d^2}{dx^2}[N]\{u\} = [B]\{u\} \Leftrightarrow [B] = \frac{d^2}{dx^2}[N] \tag{3.24}$$

by deriving the shape functions,

$$[B] = \frac{d^2}{dx^2}[N] = \left[-\frac{6}{L^2} + \frac{12x}{L^3} \quad -\frac{4}{L} + \frac{6x}{L^2} \quad \frac{6}{L^2} - \frac{12x}{L^3} \quad -\frac{2}{L} + \frac{6x}{L^2}\right] \tag{3.25}$$

finally the part of stiffness matrix related to bending is given by,

$$[K] = \int_0^L [B]^T EI_z [B]\, dx = \frac{EI_z}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix} \tag{3.26}$$

The full stiffness matrix of the Beam2D element is found by assembling (3.26) and the Truss2D stiffness matrix (3.17), as represented in figure 3.2. The stiffness matrix for the Beam2D is then:

$$[K] = \frac{E}{L^3}\begin{bmatrix} AL^2 & 0 & 0 & -AL^2 & 0 & 0 \\ 0 & 12I_z & 6LI_z & 0 & -12I_z & 6LI_z \\ 0 & 6LI_z & 4L^2I_z & 0 & -6LI_z & 2L^2I_z \\ -AL^2 & 0 & 0 & AL^2 & 0 & 0 \\ 0 & -12I_z & -6LI_z & 0 & 12I_z & -6LI_z \\ 0 & 6LI_z & 2L^2I_z & 0 & -6LI_z & 4L^2I_z \end{bmatrix} \tag{3.27}$$

The stiffness matrix for the Truss2D and Truss3D elements are exactly the same. To obtain the Beam3D matrix, however, it is necessary to consider also bending around the *y* transversal axis and torsion, which are not accounted for in Beam2D. The Beam3D uses 6 degrees of freedom per node, axial displacement, *u*, transversal displacement along *y*, *v*, transversal displacement along *z*, *w*, cross-section rotation along *x*, *y* and *z* axis, respectively θ_x, θ_y and θ_z, and the several internal energy terms are independent and related to specific degrees of freedom. As before, energy associated to axial forces is only related with *u* and energy associated to bending in *xy* plane is exclusively related with *v* and θ_z. The new terms, bending in *xz* plane and torsion are associated with *w* and θ_y and with θ_x , respectively.

The procedure to obtain the internal energy associated with bending in *xz* plane is similar to the one used before for the *xy* plane and will not be repeated. The only difference is the change of axis.

The formulation for the internal energy associated with torsion is very similar to the formulation of the truss element, changing the axial force for the twisting moment and the integral from (3.16) becomes,

$$[K] = GI_x \int_{-1}^{1} \begin{bmatrix} -1 \\ 1 \end{bmatrix} [-1 \quad 1\,]\, J\, d\xi \tag{3.28}$$

where,

$$G = \frac{E}{2(1+v)} \tag{3.29}$$

$$I_x = I_y + I_z$$

so the stiffness matrix for the θ_{x1} and θ_{x2} degree of freedom is,

$$[K] = \frac{GI_x}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{3.30}$$

finally the Beam3D stiffness matrix is obtained by assembling all the previous stiffness matrices,

$$[\,K] = \frac{E}{L^3} \times$$

$$\begin{bmatrix}
AL^2 & 0 & 0 & 0 & 0 & 0 & -AL^2 & 0 & 0 & 0 & 0 & 0 \\
 & 12I_z & 0 & 0 & 0 & 6LI_z & 0 & -12I_z & 0 & 0 & 0 & 6LI_z \\
 & & 12I_y & 0 & -6LI_y & 0 & 0 & 0 & -12I_y & 0 & 6LI_y & 0 \\
 & & & DI_xL^2 & 0 & 0 & 0 & 0 & 0 & -DI_xL^2 & 0 & 0 \\
 & & & & 4L^2I_y & 0 & 0 & 0 & 6LI_y & 0 & 2L^2I_y & 0 \\
 & & & & & 4L^2I_z & 0 & -6LI_z & 0 & 0 & 0 & 2L^2I_z \\
 & & & & & & AL^2 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & 12I_z & 0 & 0 & 0 & 6LI_z \\
 & & S & & & & & & 12I_y & 0 & 6LI_y & 0 \\
 & & & & & & & & & DI_xL^2 & 0 & 0 \\
 & & & & & & & & & & 4L^2I_y & 0 \\
 & & & & & & & & & & & 4L^2I_z
\end{bmatrix} \tag{3.31}$$

where, $D = \frac{G}{E}$ (defined just for convenience).

3.4. Coordinate System Transformation

The previous stiffness matrices were evaluated in the element local coordinate system. If the structure being simulated is composed with only one element then the coordinate system used is not an issue. However almost every structure is composed with multiple elements, after all, the larger the number of elements in which a structure is divided the better are the results, and it is necessary to assemble all the element stiffness matrices in one single matrix representing the structure. It is not possible to make this assembly of matrices if each uses her own different coordinate system and it becomes necessary to make a coordinate transformation for all these matrices so that they are expressed in a common, or global, coordinate system. This transformation, can be expressed by,

$$\left[K_g\right] = [T]^T [K_L][T] \tag{3.32}$$

where,

$\left[K_g\right]$ = element stiffness matrix in the global coordinate system.

$[K_L]$ = element stiffness matrix in the local coordinate system.

$[T]$ = transformation matrix.

Even though the procedure from (3.32) is the same for every element type, the transformation matrix $[T]$ is not. The transformation matrix holds the geometrical relationships between the local and the global coordinate system and although it is different for each type of element the differences are only significant from a 2 dimensional to a 3 dimensional element.

3.4.1. 2D Transformation Matrix

The transformation matrix for a two dimensional element is rather simple and is completely defined by the angle between the *x*-axis of the local coordinate and the *X*-axis of the global coordinate system. From this angle one can obtain the directon cosines needed for the transformation. However it is easier to use the distances between the nodes, as shown in the figure 3.3.

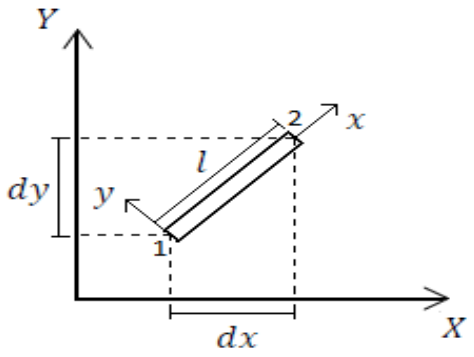

Figure 3.3 – 2D global and local coordinate system

$$\cos\alpha = \frac{dx}{l} \qquad (3.33)$$
$$\sin\alpha = \frac{dy}{l}$$

the transformation matrix for the Truss2D element is then,

$$[T] = \begin{bmatrix} cos\,\alpha & sin\,\alpha & 0 & 0 \\ 0 & 0 & cos\,\alpha & sin\,\alpha \end{bmatrix} \qquad (3.34)$$

and the transformation matrix for the Beam2D element is,

$$[T] = \begin{bmatrix} cos\,\alpha & sin\,\alpha & 0 & 0 & 0 & 0 \\ -sin\,\alpha & cos\,\alpha & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & cos\,\alpha & sin\,\alpha & 0 \\ 0 & 0 & 0 & -sin\,\alpha & cos\,\alpha & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.35)$$

3.4.2. 3D Transformation Matrix

The 3D transformation matrix involves a more substantial level of complexity. To begin, it is first necessary to define the orientation of the local coordinate system and it is possible to do so in one of two ways: (1) by the default settings as seen in fig 3.4, that will be explained later; (2) by defining the element with 3 nodes (the third node is only for orientation), as shown in the figure 3.5.

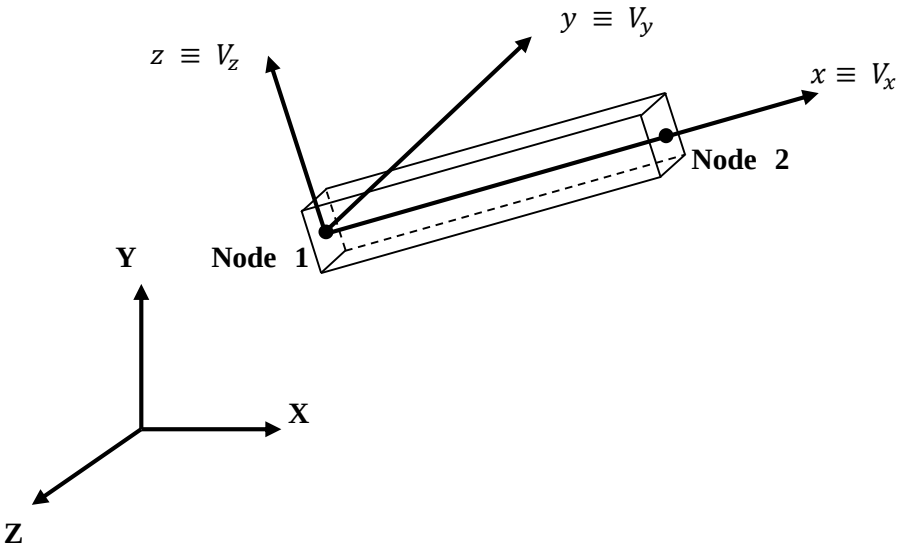

Figure 3.4 - Default orientation of the local coordinate system.

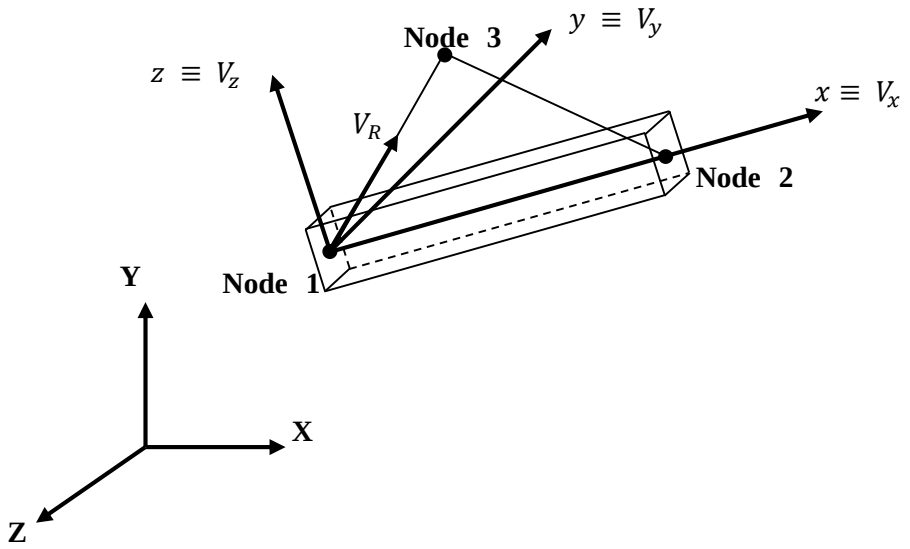

Figure 3.5 - Orientation of the local coordinate system with the third node option.

Whatever the option chosen, it doesn't affect the local x direction which is always defined by the vector passing through nodes 1 and 2,

$$V_x = V_{12} = \begin{Bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{Bmatrix} = \begin{Bmatrix} x_{21} \\ y_{21} \\ z_{21} \end{Bmatrix} \tag{3.36}$$

the direction cosine λ_x is obtained by normalizing the V_x vector,

$$\lambda_x = \frac{1}{l_{21}} \begin{Bmatrix} x_{21} \\ y_{21} \\ z_{21} \end{Bmatrix} \tag{3.37}$$

where l_{21} is the length of the element,

$$l_{21} = \sqrt{{x_{21}}^2 + {y_{21}}^2 + {z_{21}}^2} \tag{3.38}$$

The option chosen regarding the orientation of the system only becomes relevant for the local *y* direction. If a third node has been defined then the procedure is the following,

$$V_r = V_{13} = \begin{Bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \end{Bmatrix} = \begin{Bmatrix} x_{31} \\ y_{31} \\ z_{31} \end{Bmatrix} \tag{3.39}$$

The V_r is an auxiliary vector used to define a plane alongside with V_x vector, from which the local *y* direction is perpendicular. To obtain the a vector perpendicular to a plane composed by 2 other vector, the external product is used,

$$V_y = V_r \times V_x \tag{3.40}$$

On the other hand if the third node was not defined, then by default EFFECT automatically calculates the V_y vector to be parallel to the *X*-*Y* plane on the global coordinate system. Finally the local *z* direction is given by the external product of V_x and V_y,

$$V_z = V_x \times V_y \tag{3.41}$$

The directions cosine λ_y and λ_z are obtained by normalizing the vector V_y and V_z respectively. The 3×3 transformation matrix can now be written as,

$$[\lambda]_{3\times3} = \begin{bmatrix}\{\lambda_x\} & \{\lambda_y\} & \{\lambda_z\}\end{bmatrix} = \begin{bmatrix}(\lambda_x)_1 & (\lambda_y)_1 & (\lambda_z)_1 \\ (\lambda_x)_2 & (\lambda_y)_2 & (\lambda_z)_2 \\ (\lambda_x)_3 & (\lambda_y)_3 & (\lambda_z)_3\end{bmatrix} \tag{3.42}$$

The transformation matrices of the Truss3D and Beam3D can be obtained by repeating the matrix (3.42), as it is going to be shown. The Truss3D transformation matrix is then,

$$[T] = \begin{bmatrix}[\lambda]_{3\times3} & [0]_{3\times3} \\ [0]_{3\times3} & [\lambda]_{3\times3}\end{bmatrix} \tag{3.43}$$

and the Beam3D transformation matrix is,

$$[T] = \begin{bmatrix}[\lambda]_{3\times3} & [0]_{3\times3} & [0]_{3\times3} & [0]_{3\times3} \\ [0]_{3\times3} & [\lambda]_{3\times3} & [0]_{3\times3} & [0]_{3\times3} \\ [0]_{3\times3} & [0]_{3\times3} & [\lambda]_{3\times3} & [0]_{3\times3} \\ [0]_{3\times3} & [0]_{3\times3} & [0]_{3\times3} & [\lambda]_{3\times3}\end{bmatrix} \tag{3.44}$$

Chapter 4

4. Triangular Flat Shell Element

4.1. Overview

The triangular flat shell is formed by superimposing the Constant Stain Triangle (CST) element expressing the membrane behavior and the Discrete Kirchhoff Theory (DKT) element developed by Batoz *et al.* [2] for the plate bending behavior. The assembly of those 2 elements results in a flat shell element with 18 degrees of freedom, 6 for each node:

$$\{U\}^T = \{u \quad v \quad w \quad \theta_x \quad \theta_y \quad \theta_z\}$$

the CST element is associated with 2 of these degrees of freedom:

$$\{U_m\}^T = \{u \quad v\}$$

and the DKT element with 3:

$$\{U_b\}^T = \{w \quad \theta_x \quad \theta_y\}$$

The rotation about the local z-axis (θ_z), represented by a fictitious value, is discussed further in the sub-chapter 4.4. This chapter presents the formulation of the stiffness matrix for each part and finally the overall stiffness matrix for the triangular flat element. The assembly and the degrees of freedom are represented in the figure 4.1 and 4.2.

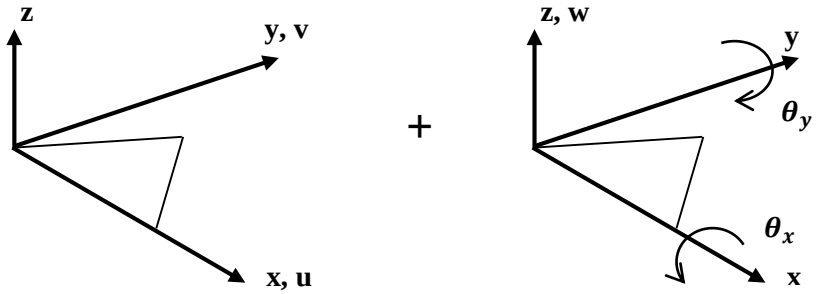

Figure 4.1 - CST element on the left and the DKT element on the right, with their associated degrees of freedom.

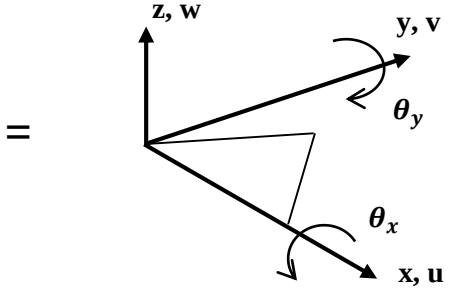

Figure 4.2 - Assembly of the CST element with the DKT element.

4.2. CST Element Stiffness Matrix Formulation

The formulation for obtaining the CST element stiffness matrix is presented in [9]. The displacements can be obtained by interpolation,

$$u = [1 \quad x \quad y]\begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix} \qquad v = [1 \quad x \quad y]\begin{Bmatrix} a_4 \\ a_5 \\ a_6 \end{Bmatrix} \tag{4.1}$$

or,

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{Bmatrix} \quad or \quad \{U(x,y)\} = [X]\{a\} \tag{4.2}$$

Where $\{U(x,y)\}$ now only contains the membrane part of the displacements. The stains are obtained through the derivations,

$$\varepsilon_x = \frac{du}{dx} = a_2$$

$$\varepsilon_y = \frac{dv}{dy} = a_6 \tag{4.3}$$

$$\gamma_{xy} = \frac{du}{dy} + \frac{dv}{dx} = a_3 + a_5$$

Because the displacement functions are linear, the element stains, as can be seen in equation (4.3), are constant, so the name "constant stain triangle" (CST). Evaluating the expression (4.1) at all the three nodes,

$$\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} = [A] \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{Bmatrix} \quad or, \quad \{u\} = [A]\{a\} \tag{4.4}$$

where,

$$[A] = \begin{bmatrix} 1 & x_1 & y_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_1 & y_1 \\ 1 & x_2 & y_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_2 & y_2 \\ 1 & x_3 & y_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_3 & y_3 \end{bmatrix} \tag{4.5}$$

Reversing (4.4),

$$\{a\} = [A]^{-1}\ \{u\} \tag{4.6}$$

and combining with equation (4.2) **,** one obtains,

$$\{U(x,y)\} = [X][A]^{-1}\{u\} \tag{4.7}$$

where the shape functions $[N]$ are,

$$[N] = [X][A]^{-1} \tag{4.8}$$

and where,

$$[\mathrm{A}]^{-1} = \frac{1}{2\mathrm{A}} \times$$

$$\times \begin{bmatrix} x_2y_3 - x_3y_2 & 0 & x_3y_1 - x_1y_3 & 0 & x_1y_2 - x_2y_1 & 0 \\ y_2 - y_3 & 0 & y_3 - y_2 & 0 & y_1 - y_2 & 0 \\ x_3 - x_2 & 0 & x_1 - x_2 & 0 & x_2 - x_1 & 0 \\ 0 & x_2y_3 - x_3y_2 & 0 & x_3y_1 - x_1y_3 & 0 & x_1y_2 - x_2y_1 \\ 0 & y_2 - y_3 & 0 & y_3 - y_2 & 0 & y_1 - y_2 \\ 0 & x_3 - x_2 & 0 & x_1 - x_2 & 0 & x_2 - x_1 \end{bmatrix}$$

(4. 9)

and *A* is the area of the element,

$$A = \frac{1}{2}[x_1\ (y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)] \tag{4.10}$$

The 3 shape functions are then obtained from the equation (4. 8):

$$N_1 = \frac{1}{2A}[(x_2y_3\ -\ x_3y_2) - (y_2\ - y_3)x + (x_3\ -\ x_2)y]$$

$$N_2 = \frac{1}{2A}[(x_3y_1\ -\ x_1y_3) - (y_3\ - y_2)x + (x_1\ -\ x_2)y] \tag{4.11}$$

$$N_3 = \frac{1}{2A}[(x_1y_2\ -\ x_2y_1) - (y_1\ - y_2)x + (x_2\ -\ x_1)y]$$

the relationship between the stains and the displacements is,

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = [B] \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{4.12}$$

where $[B]$ it is the strain-displacement matrix,

$$[B] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial y} \end{bmatrix} \tag{4.13}$$

deriving (4. 13) the following expression for [B] is obtained:

$$[B] = \frac{1}{2A}\begin{bmatrix} y_2 - y_3 & 0 & y_3 - y_2 & 0 & y_1 - y_2 & 0 \\ 0 & x_3 - x_2 & 0 & x_1 - x_3 & 0 & x_2 - x_1 \\ x_3 - x_2 & y_2 - y_3 & x_1 - x_3 & y_3 - y_1 & x_2 - x_1 & y_1 - y_2 \end{bmatrix} \tag{4.14}$$

the stiffness matrix of the CST element is given by the following equation,

$$[K] = \int_V [B]^T [E] [B] dV \tag{4.15}$$

for constant thickness, the volume integral can be obtained multiplying the thickness by an area integral,

$$[K] = t \int_A [B]^T [E] [B] \ dA \tag{4.16}$$

where *t* is the thickness of the element, $[B]$ is the strain matrix obtained in (4.14) and $[E]$ is the constitutive matrix for plane stress,

$$[E] = \frac{E}{1-\upsilon^2}\begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & (1-\upsilon)/2 \end{bmatrix} \tag{4.17}$$

where,

E = modulus of elasticity of the material.
υ = Poisson's coefficient.

Because the thickness and the constitutive matrix are constant in equation (4.16) one obtains,

$$[K] = [B]^T [E] [B] \ t \ A \tag{4.18}$$

4.3. DKT Element Stiffness Matrix Formulation

Batoz *et al.* [2] based the formulation of this element on the thick plate Mindlin theory and used a generalization of the Kirchhoff hypothesis, imposed at discrete locations within the element: "points of the plate originally on the normal to the undeformed middle surface remain on a straight line, which is normal to the deformed middle surface."

According with the above assumption, the displacement components *u, v* and *w* at any point with coordinates *x, y, z* can be represented as,

$$u = z\beta_x(x,y); \qquad v = z\beta_y(x,y); \qquad w = w(x,y) \tag{4.19}$$

Where, w is the transverse displacement, and β_x and β_y are the rotations in the direction normal to the x-z and y-z planes respectively, as shown in figure 4.3.

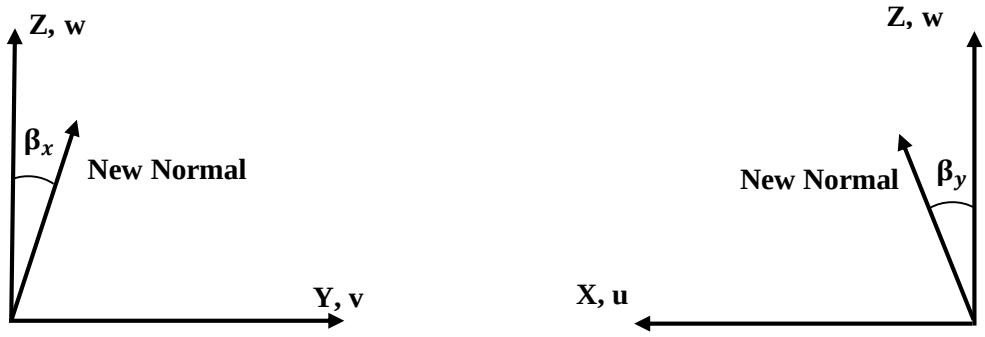

Figure 4.3 - Positive direction of β_x and β_y.

According to Batoz et al. [2], for thin plates the transverse shear strain energy is negligible compared to the bending energy and so the stiffness matrix of the DKT element is based on the expression,

$$U = \frac{1}{2}\int_A [k]^T [D_b]\,[k]\; dx\, dy \tag{4.20}$$

where,

$$[D_b] = \frac{E\,t^3}{12(1-\upsilon)}\begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & (1-\upsilon)/2 \end{bmatrix} \tag{4.21}$$

and,

E = modulus of elasticity of the material.

υ = Poisson's coefficient.

t = thickness of the element.

and the curvature, $[k]$, is obtained with,

$$[k] = \begin{bmatrix} \beta_{x,x} \\ \beta_{y,y} \\ \beta_{x,y} + \beta_{y,x} \end{bmatrix} \tag{4.22}$$

Batoz *et al.* [2] made the following assumptions for the formulation of the DKT element:

β_x and β_y vary quadratically over the element, i.e.

$$\beta_x = \sum_{i=1}^{6} N_i \beta_{x_i} \qquad \beta_y = \sum_{i=1}^{6} N_i \beta_{y_i} \tag{4.23}$$

where β_{x_i} and β_{y_i} are the nodal values at the corners and at the mid-nodes. They use the following $N_i(\xi, \eta)$ shape functions,

$$\begin{aligned} N_1 &= 2(1 - \xi - \eta)\left(\frac{1}{2} - \xi - \eta\right) \\ N_2 &= \xi(2\xi - 1) \\ N_3 &= \eta(2\eta - 1) \\ N_4 &= 4\xi\eta \\ N_5 &= 4\eta(1 - \xi - \eta) \\ N_6 &= 4\xi(1 - \xi - \eta) \end{aligned} \tag{4.24}$$

where (ξ, η) are the natural coordinates of the element as shown in figure 4.4.

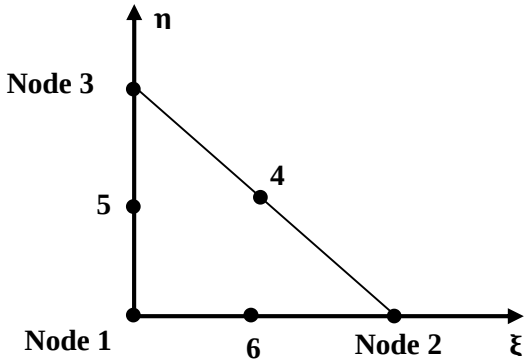

Figure 4.4 - Natural coordinates of the DKT Element.

The Kirchhoff hypothesis is imposed at:

The corner nodes,

$$\gamma = \begin{bmatrix} \beta_x + w_{,x} \\ \beta_y + w_{,y} \end{bmatrix} = 0 \qquad at\ nodes\ 1, 2\ and\ 3 \tag{4.25}$$

The mid-nodes, along the direction *s* aligned with the element edge,

$$\beta_{s_k} + w_{,s_k} \quad k = 4, 5, 6 \tag{4.26}$$

The variation of *w* along the sides is cubic, i.e.

$$w_{,s_k} = -\frac{3}{2l_{ij}}w_i - \frac{1}{4}w_{,s_i} + \frac{3}{2l_{ij}}w_j - \frac{1}{4}w_{,s_j} \tag{4.27}$$

with *k* denoting the mid-node of side *ij* and l_{ij} equal to the length of side *ij*.

A linear variation of β_n is imposed along the side, i.e.

$$\beta_{n_k} = \frac{1}{2}\left(\beta_{n_i} + \beta_{n_j}\right) \tag{4.28}$$

where *k = 4,5,6* denotes the mid-node of sides *2-3,3-1* and *1-2*, respectively and n_k points in the directions normal to the element edges.

Using the expression (4.25) and the geometrical relations,

$$\begin{bmatrix} w_{,s} \\ w_{,n} \end{bmatrix} = \begin{bmatrix} c & s \\ s & -c \end{bmatrix} \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix} \tag{4.29}$$

where,

$$c = -\frac{y_{ij}}{l_{ij}} \qquad\qquad s = \frac{y_{ij}}{l_{ij}} \tag{4.30}$$

and,

$$\begin{aligned} y_{ij} &= y_i - y_j \\ x_{ij} &= x_i - x_j \\ l_{ij} &= \sqrt{{x_{ij}}^2 + {y_{ij}}^2} \end{aligned} \tag{4.31}$$

with *i* and *j* denoting different nodes.

The β_x and β_y are obtained in terms of the displacements,

$$\begin{aligned} \beta_x &= [H_x^T(\xi,\eta)]\{U\} \\ \beta_y &= \left[H_y^T(\xi,\eta)\right]\{U\} \end{aligned} \tag{4.32}$$

where $\{U\}$ contains the 9 displacements at the nodes, associated with bending,

$$\{U\}^T = \left\{w_1 \quad \theta_{x_1} \quad \theta_{y_1} \quad w_2 \quad \theta_{x_2} \quad \theta_{y_2} \quad w_3 \quad \theta_{x_3} \quad \theta_{y_3}\right\}$$

and the[H_x] and $\left[H_y \right]$, are the nine component vectors of the new shape functions,

$$[H^x(\xi,\eta)] = \begin{bmatrix} 1.5(\ a_6N_6 - a_5N_5\ ) \\ b_5N_5 + b_6N_6 \\ N_1 - \ c_5N_5 - c_6N_6 \\ 1.5(\ a_4N_4 - a_6N_6\ ) \\ b_6N_6 + b_4N_4 \\ N_2 - \ c_6N_6 - c_4N_4 \\ 1.5(\ a_5N_5 - a_4N_4\ ) \\ b_4N_4 + b_5N_5 \\ N_3 - \ c_4N_4 - c_5N_5 \end{bmatrix} \tag{4.33}$$

and the H_y component vectors are,

$$[H^x(\xi,\eta)] = \begin{bmatrix} 1.5(\ d_6N_6 - d_5N_5\ ) \\ -\ N_1 + \ e_5N_5 + e_6N_6 \\ -\ b_5N_5 - b_6N_6 \\ 1.5(\ d_4N_4 - d_6N_6\ ) \\ -\ N_2 + \ e_6N_6 + e_4N_4 \\ -\ b_6N_6 - b_4N_4 \\ 1.5(\ d_5N_5 - d_4N_4\ ) \\ -\ N_3 + \ e_4N_4 + e_5N_5 \\ -\ b_4N_4 - b_5N_5 \end{bmatrix} \tag{4.34}$$

where,

$$a_k = -\frac{x_{ij}}{{l_{ij}}^2}$$

$$b_k = -\frac{3}{4}\frac{x_{ij}\ y_{ij}}{{l_{ij}}^2}$$

$$c_k = \left(\frac{1}{4}{x_{ij}}^{\ 2} - \frac{1}{2}{y_{ij}}^{\ 2}\right)/{l_{ij}}^2 \tag{4.35}$$

$$d_k = -\frac{y_{ij}}{{l_{ij}}^2}$$

$$e_k = \left(\frac{1}{4}{y_{ij}}^{\ 2} - \frac{1}{2}{x_{ij}}^{\ 2}\right)/{l_{ij}}^2$$

$${l_{ij}}^2 = \left(\ {x_{ij}}^{\ 2} + \ {y_{ij}}^{\ 2}\right)$$

where *k = 4, 5, 6* for the sides *ij = 2-3, 3-1, 1-2* respectively.

The stain-displacement transformation matrix is,

$$[B(\xi,\eta)\,] = \frac{1}{2A}\begin{bmatrix} y_{31}\left[H_{x,\xi}\right]^T + y_{12}\left[H_{x,\eta}\right]^T \\ -x_{31}\left[H_{y,\xi}\right]^T - x_{12}\left[H_{y,\eta}\right]^T \\ -x_{31}\left[H_{x,\xi}\right]^T - x_{12}\left[H_{x,\eta}\right]^T + y_{31}\left[H_{y,\xi}\right]^T + y_{12}\left[H_{y,\eta}\right]^T \end{bmatrix} \tag{4.36}$$

and,

$$2A = x_{31}y_{12} - x_{12}y_{31} \tag{4.37}$$

where the derivatives of the component vectors of the shape functions can be represented as following:

Derivatives of the equations (4.33) and (4.34) with respect to ξ,

$$\left[H_{x,\xi}\right] = \begin{bmatrix} P_6(1-2\xi) + (P_5 - P_6)\eta \\ q_6(1-2\xi) - (q_5 + q_6)\eta \\ -4 + 6(\xi+\eta) + r_6(1-2\xi) - \eta(r_5 + r_6) \\ -P_6(1-2\xi) + \eta(P_4 + P_6) \\ q_6(1-2\xi) - \eta(q_6 - q_4) \\ -2 + 6\xi + r_6(1-2\xi) - \eta(r_4 - r_6) \\ -\eta(P_5 + P_4) \\ \eta(q_4 - q_5) \\ -\eta(r_5 - r_4) \end{bmatrix} \tag{4. 38}$$

$$\left[H_{y,\xi}\right] = \begin{bmatrix} t_6(1-2\xi) + \eta(t_5 - t_6) \\ 1 + r_6(1-2\xi) - \eta(r_5 + r_6) \\ q_6(1-2\xi) + \eta(q_5 + q_6) \\ -t_6(1-2\xi) + \eta(t_4 + t_6) \\ -1 + r_6(1-2\xi) + \eta(r_4 - r_6) \\ -q_6(1-2\xi) - \eta(q_4 - q_6) \\ -\eta(t_4 + t_5) \\ \eta(r_4 - r_5) \\ -\eta(q_4 - q_5) \end{bmatrix} \tag{4. 39}$$

Derivatives of the equations (4.36) and (4.37) with respect to η,

$$\left[H_{x,\eta}\right] = \begin{bmatrix} -P_5(1-2\eta) - \xi(P_6 - P_5) \\ q_5(1-2\eta) - \xi(q_5 + q_6) \\ -4 + 6(\xi+\eta) + r_5(1-2\eta) - \xi(r_5 + r_6) \\ \xi(P_4 + P_6) \\ \xi(q_4 - q_6) \\ -\xi(r_6 - r_4) \\ P_5(1-2\eta) - \xi(P_4 + P_5) \\ q_5(1-2\eta) + \xi(q_4 - q_5) \\ -2 + 6\eta + r_5(1-2\eta) + \xi(r_4 - r_6) \end{bmatrix} \tag{4. 40}$$

$$\left[H_{y,\xi}\right] = \begin{bmatrix} -t_5(1-2\eta) - \xi(t_6 - t_5) \\ 1 + r_5(1-2\eta) - \xi(r_5 + r_6) \\ q_5(1-2\eta) + \xi(q_5 + q_6) \\ \xi(t_4 + t_6) \\ \xi(r_4 - r_6) \\ -\xi(q_4 - q_6) \\ t_5(1-2\eta) - \xi(t_4 + t_5) \\ -1 + r_5(1-2\eta) - \xi(r_4 - r_5) \\ q_5(1-2\eta) - \xi(q_4 - q_5) \end{bmatrix} \tag{4.41}$$

where,

$$P_k = -\frac{6x_{ij}}{l_{ij}^2}$$

$$q_k = \frac{3x_{ij}y_{ij}}{l_{ij}^2} \tag{4.42}$$

$$r_k = \frac{3{y_{ij}}^2}{l_{ij}^2}$$

$$t_k = -\frac{6y_{ij}}{l_{ij}^2}$$

and, *k =4, 5, 6* for *ij = 2-3, 3-2, 1-2* respectively.

The stiffness matrix of the DKT element is obtained by solving the following integral,

$$[K] = 2A \int_0^1 \int_0^{1-\eta} [B]^T [D_b][B] \, d\xi d\eta \tag{4.43}$$

According to Batoz *et al.* [2] it is appropriate to use a three point numerical integration scheme, with points located at the mid-nodes. Assuming the thickness and the material properties are constant over the element, the exact integration of the stiffness matrix gives,

$$[K] = 2A \sum_{j=1}^{3} \sum_{i=1}^{3} w_i w_j \, [B]^T [D_b][B] \tag{4.44}$$

The coordinates and the weight functions for the integration points are as following,

Table 4.1- Coordinates and weight functions for Gauss quadrature.

Integration point	Coordinates	Weight functions
1	$\left(\frac{1}{2}, 0\right)$	$1/3$
2	$\left(\frac{1}{2}, \frac{1}{2}\right)$	$1/3$
3	$\left(0, \frac{1}{2}\right)$	$1/3$

4.4. Flat shell element stiffness matrix formulation

After formulating the CST and DKT stiffness matrices it is now possible to obtain the stiffness matrix of the triangular flat element by assembling. However, the combination of the 2 elements only involves 5 degrees of freedom on each node, 2 for the CST and 3 for the DKT. The sixth degree of freedom, rotation about the local z-axis (θ_z) is called the drilling degree of freedom and remains undetermined, i.e., there are no energy values associated to it.

The approach chosen to deal with the drilling degree of freedom is to introduce a fictitious stiffness value. This approach sometimes results in a stiffer structure due to the constraints present at the corner nodes but it is the easier approach to implement and gives satisfactory results.

The stiffness matrix of the triangular flat shell element at each node can then be represented by:

$$\left[K_{tfs}\right]_i = \begin{bmatrix} [K_m]_{2\times2} & [0]_{2\times3} & 0 \\ [0]_{3\times2} & [K_b]_{3\times3} & 0 \\ 0 & 0 & f \end{bmatrix} \tag{4.45}$$

where,

$\left[K_{tfs}\right]_i$ = stiffness matrix at each node of the triangular flat shell element.

$[K_m]_{2\times2}$ = CST stiffness matrix at each node of the shell element.

$[K_b]_{3\times3}$ = DKT stiffness matrix at each node of the shell element.

f = Fictitious stiffness value of max $\left(\frac{(Ktfs)_{i,i}}{1000}\right)$

4.5. Coordinate Transformation

The process for the coordinate transformation is similar to the one used in the previous chapter,

$$\left[K_g\right] = [T]^T [K_L][T] \tag{4.46}$$

where,

$\left[K_g\right]$ = element stiffness matrix in the global coordinate system.

$[K_L]$ = element stiffness matrix in the local coordinate system.

$[T]$ = transformation matrix.

Again it is first necessary to define the orientation of the local coordinate system, in order to construct the transformation matrix. The local coordinate system, for the triangular flat shell element is shown in figure 4.5, and the procedure to obtain the transformation matrix, is very similar to the one used for the Beam3D on the previous chapter.

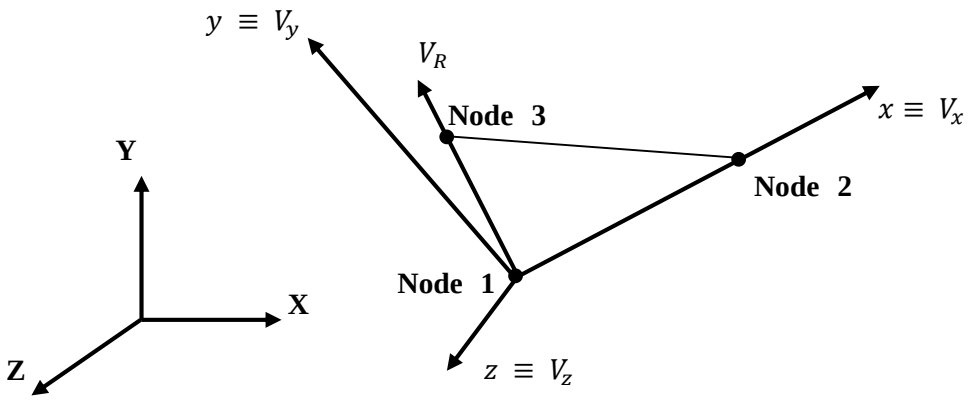

Figure 4.5 - Orientation of the local coordinate system.

The local *x* direction is given by the vector passing through nodes 1 and 2,

$$V_x = V_{12} = \begin{Bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{Bmatrix} = \begin{Bmatrix} x_{21} \\ y_{21} \\ z_{21} \end{Bmatrix} \tag{4.47}$$

the direction cosine λ_x is obtained by normalizing the V_x vector,

$$\lambda_x = \frac{1}{l_{21}} \begin{Bmatrix} x_{21} \\ y_{21} \\ z_{21} \end{Bmatrix} \tag{4.48}$$

where l_{21} is the length of the side of the element,

$$l_{21} = \sqrt{{x_{21}}^2 + {y_{21}}^2 + {z_{21}}^2} \tag{4.49}$$

for the local *y* direction it is necessary to define an auxiliary vector,

$$V_r = V_{13} = \begin{Bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \end{Bmatrix} = \begin{Bmatrix} x_{31} \\ y_{31} \\ z_{31} \end{Bmatrix} \tag{4.50}$$

Vector V_r is used with vector V_x to define a plane from which the local *z* direction is perpendicular. To obtain the vector perpendicular to a plane composed by 2 other vector, the external product is used,

$$V_z = V_x \times V_r \tag{4.51}$$

finally the local *y* direction is given by external product of V_z and V_x,

$$V_y = V_z \times V_x \tag{4.52}$$

The directions cosine λ_y and λ_z are obtained by normalizing the vector V_y and V_z respectively. The 3×3 transformation matrix can now be written as,

$$[\lambda]_{3\times3} = \begin{bmatrix} \{\lambda_x\} & \{\lambda_y\} & \{\lambda_z\} \end{bmatrix} = \begin{bmatrix} (\lambda_x)_1 & (\lambda_y)_1 & (\lambda_z)_1 \\ (\lambda_x)_2 & (\lambda_y)_2 & (\lambda_z)_2 \\ (\lambda_x)_3 & (\lambda_y)_3 & (\lambda_z)_3 \end{bmatrix} \tag{4.53}$$

and,

$$[T_p]_{6\times6} = \begin{bmatrix} [\lambda]_{3\times3} & [0]_{3\times3} \\ [0]_{3\times3} & [\lambda]_{3\times3} \end{bmatrix} \tag{4.54}$$

The transformation matrix can be obtained by repeating the matrix (4.54),

$$[T]_{18\times18} = \begin{bmatrix} [T_p] & [0] & [0] \\ [0] & [T_p] & [0] \\ [0] & [0] & [T_p] \end{bmatrix} \tag{4.55}$$

Chapter 5

5. Quadrilateral flat shell element

5.1. Overview

The quadrilateral flat shell, as the triangular flat shell, is composed from two different elements, the isoparametric four node quadrilateral element (Q4) to model the membrane behavior and the Discrete Kirchoff Quadrilateral (DKQ) element to represent the plate bending behavior. The quadrilateral flat shell element has a total of 24 degrees of freedom, 6 for each node:

$$\{U\}^T = \{u \quad v \quad w \quad \theta_x \quad \theta_y \quad \theta_z\}$$

the Q4 element is associated with 2 of these degrees of freedom:

$$\{U_m\}^T = \{u \quad v\}$$

and the DKQ element with 3:

$$\{U_b\}^T = \{w \quad \theta_x \quad \theta_y\}$$

For the drilling degree of freedom it was once again introduced a fictitious value. This chapter is intended to show the formulation of the stiffness matrix for the Q4 and DKQ elements and finally the overall stiffness matrix for the quadrilateral flat shel. The assembly and the degrees of freedom are represented in the figure 5.1 and 5.2.

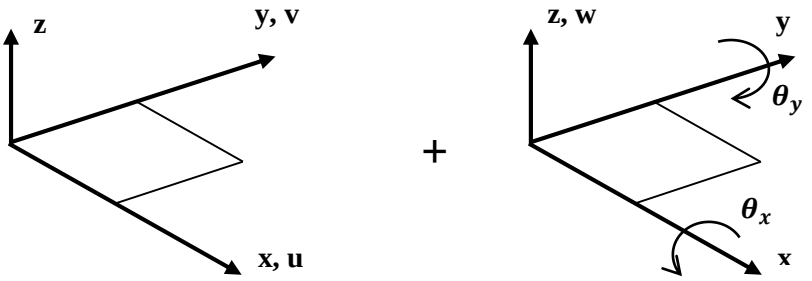

Figure 5.1 – Q4 element on the left and the DKQ element on the Right, with their respectively degrees of freedom.

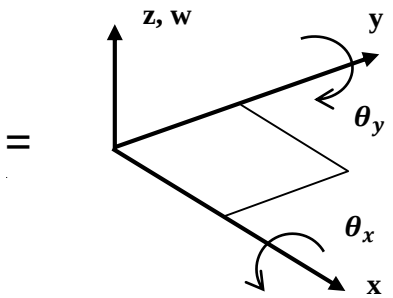

Figure 5.2 – Assembly of the Q4 element with the DKQ element.

5.2. Q4 Element Stiffness Matrix Formulation

The isoparametric four node quadrilateral element, like all isoparametric elements, need a set of auxiliary natural coordinates (ξ, η), as the element is two dimensional as shown in figure 5.3.

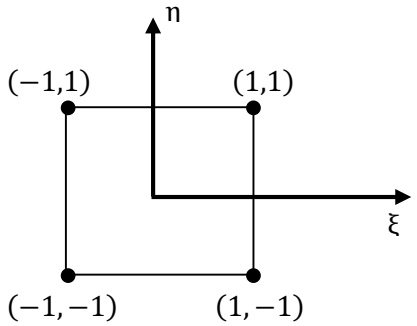

Figure 5.3 - Natural coordinates of the Q4 Element.

The formulation of the Q4 element stiffness matrix, according to [9], is now presented. Vector $\{U\}$ now only contains de nodal displacements related to membrane behavior, i.e., equals $\{U_m\}$.

The coordinates at a point within the element (natural coordinates) can be obtained by interpolation of the coordinates on the global system:

$$\begin{Bmatrix} x(\xi,\eta) \\ y(\xi,\eta) \end{Bmatrix} = \begin{Bmatrix} \sum N_i x_i \\ \sum N_i y_i \end{Bmatrix} = [N]\{c\} \tag{5.1}$$

where,

$$[N] = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{bmatrix} \tag{5.2}$$

$$\{c\} = \{x_1 \quad y_1 \quad x_2 \quad y_2 \quad x_3 \quad y_3 \quad x_4 \quad y_4\}^T$$

the same shape functions can be used to interpolate the displacements,

$$\begin{Bmatrix} u(\xi,\eta) \\ v(\xi,\eta) \end{Bmatrix} = \begin{Bmatrix} \sum N_i u_i \\ \sum N_i v_i \end{Bmatrix} = [N]\{U\} \tag{5.3}$$

where,

$$\{U\} = \{u_1 \quad v_1 \quad u_2 \quad v_2 \quad u_3 \quad v_3 \quad u_4 \quad v_4\}^T \tag{5.4}$$

the individual shape functions are:

$$N_1 = \frac{1}{4}(1 - \xi)(1 - \eta)$$

$$N_2 = \frac{1}{4}(1 + \xi)(1 - \eta) \tag{5.5}$$

$$N_3 = \frac{1}{4}(1 + \xi)(1 + \eta)$$

$$N_4 = \frac{1}{4}(1 - \xi)(1 + \eta)$$

the strain-displacement relationships are as usual,

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \dfrac{du}{dx} \\ \dfrac{dv}{dy} \\ \dfrac{du}{dy} + \dfrac{dv}{dx} \end{Bmatrix} \tag{5.6}$$

Because the geometry on isoparametric elements is defined in the natural coordinate system, the usual derivatives in equation (5.6) are not available directly, it is necessary to use the chain rule of differentiation,

$$\frac{\partial u}{\partial \xi} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial \xi} \qquad \frac{\partial v}{\partial \xi} = \frac{\partial v}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial v}{\partial y}\frac{\partial y}{\partial \xi} \tag{5.7}$$

$$\frac{\partial u}{\partial \eta} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial \eta} \qquad \frac{\partial v}{\partial \eta} = \frac{\partial v}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial v}{\partial y}\frac{\partial y}{\partial \eta} \tag{5.8}$$

or,

$$\begin{Bmatrix} \dfrac{\partial u}{\partial \xi} \\ \dfrac{\partial u}{\partial \eta} \end{Bmatrix} = [J] \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial y} \end{Bmatrix} \qquad \begin{Bmatrix} \dfrac{\partial v}{\partial \xi} \\ \dfrac{\partial v}{\partial \eta} \end{Bmatrix} = [J] \begin{Bmatrix} \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial y} \end{Bmatrix} \tag{5.9}$$

where $[J]$ is the Jacobian matrix,

$$[J] = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{5.10}$$

where,

$$J_{11} = \frac{\partial x}{\partial \xi} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} x_i$$

$$J_{12} = \frac{\partial x}{\partial \xi} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} y_i$$

$$J_{21} = \frac{\partial y}{\partial \eta} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} x_i \qquad (5.11)$$

$$J_{22} = \frac{\partial y}{\partial \eta} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} y_i$$

by inverting the Jacobian matrix is now possible to obtain the derivatives needed in equation (5.6),

$$\begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \end{Bmatrix} \qquad \begin{Bmatrix} \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{Bmatrix} \qquad (5.12)$$

or,

$$\begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \end{Bmatrix} = \frac{1}{|J|} \begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ -J_{21} & J_{11} & 0 & 0 \\ 0 & 0 & J_{22} & -J_{12} \\ 0 & 0 & -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{Bmatrix} \qquad (5.13)$$

where,

$$[J]^{-1} = \frac{1}{|J|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \qquad (5.14)$$

and,

$$|J| = det[J] = J_{11}J_{22} - J_{21}J_{12} \qquad (5.15)$$

the strain-displacement relationships can be state as,

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{Bmatrix} \frac{du}{dx} \\ \frac{du}{dy} \\ \frac{dv}{dx} \\ \frac{dv}{dy} \end{Bmatrix} \qquad (5.16)$$

using (5.13) and (5.16) it is possible to obtain,

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \frac{1}{|J|}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}\begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ -J_{21} & J_{11} & 0 & 0 \\ 0 & 0 & J_{22} & -J_{12} \\ 0 & 0 & -J_{21} & J_{11} \end{bmatrix}\begin{Bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{Bmatrix} \quad \Leftrightarrow$$

$$\Leftrightarrow \{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \frac{1}{|J|}\begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ 0 & 0 & -J_{21} & J_{11} \\ -J_{21} & J_{11} & J_{22} & -J_{12} \end{bmatrix}\begin{Bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{Bmatrix} \tag{5.17}$$

Derivatives of displacements with respect to natural coordinates (ξ, η) can be computed with,

$$\begin{Bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} & 0 \\ \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} & 0 \\ 0 & \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} \\ 0 & \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} \end{bmatrix}\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{Bmatrix} \tag{5.18}$$

Using (5.17) and (5.18) we obtain,

$$\{\varepsilon\} = [B]\{u\} \tag{5.19}$$

where the strain-displacement matrix $[B]$ is given by,

$$[B] = \frac{1}{|J|}\begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ 0 & 0 & -J_{21} & J_{11} \\ -J_{21} & J_{11} & J_{22} & -J_{12} \end{bmatrix}\begin{bmatrix} \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} & 0 \\ \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} & 0 \\ 0 & \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} \\ 0 & \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} \end{bmatrix} \tag{5.20}$$

the stiffness matrix of the Q4 element is given by the following expression,

$$[K] = \int_V [B]^T [D] [B] dV \tag{5.21}$$

for constant thickness, the volume integral can be obtained multiplying the thickness by an area integral,

$$[K] = t \iint [B]^T [D] [B] \; dx \, dy = [K] = t \int_{-1}^{1} \int_{-1}^{1} [B]^T [D] [B] \; d\xi \, d\eta \tag{5.22}$$

where *t* is the thickness of the element, the $[B]$ is the strain matrix obtained in (5.20) and $[D]$ is the constitutive matrix for plane stress state,

$$[D] = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix} \tag{5.23}$$

5.3. DKQ Element Stiffness Matrix Formulation

The DKQ element was developed by Batoz and Tahar [3], and, likewise the DKT element, was based on the Kirchoff assumptions, i.e., the shear strain energy is neglected. The formulation of the stiffness matrix of the DKQ element is based on the following equation,

$$U = \sum_e U_b^e \tag{5.24}$$

where U_b^e is the element strain energy due to bending and is given by,

$$U = \frac{1}{2} \int_A [k]^T [D_b] [k] \; dx \, dy \tag{5.25}$$

where $[D_b]$ is the constitutive matrix relating bending moments and forces with curvatures $[k]$,

$$[D_b] = \frac{Et^3}{12(1-\upsilon^2)} \begin{bmatrix} 1 & \nu & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & (1-\upsilon)/2 \end{bmatrix} \tag{5.26}$$

and,

E = modulus of elasticity of the material.
υ = Poisson's coefficient.
t = thickness of the element.

and the curvatures $[k]$, are given by,

$$[k] = \begin{bmatrix} \beta_{x,x} \\ \beta_{y,y} \\ \beta_{x,y} + \beta_{y,x} \end{bmatrix} \tag{5.27}$$

Batoz and Tahar [3] made the following assumptions for the formulation of the DKQ element:

(1) β_x and β_y vary quadratically over the element, i.e.

$$\beta_x = \sum_{i=1}^{8} N_i \beta_{x_i} \qquad \beta_y = \sum_{i=1}^{8} N_i \beta_{y_i} \tag{5.28}$$

where the $N_i(\xi, \eta)$ are the following shape functions,

$$\begin{aligned}
\mathrm{N}_1 &= -\frac{1}{4}\,[(1-\xi)\,(1-\eta)(1+\xi+\eta)] \\
N_2 &= -\frac{1}{4}\,[(1+\xi)(1-\eta)(1-\xi+\eta)] \\
\mathrm{N}_3 &= -\frac{1}{4}\,[(1+\xi)\,(1+\eta)(1-\xi-\eta)] \\
N_4 &= -\frac{1}{4}\,[(1-\xi)(1+\eta)(1+\xi-\eta)] \\
N_5 &= \frac{1}{2}\,(1+\xi)(1-\eta^2) \\
N_6 &= \frac{1}{2}\,(1-\xi^2)(1-\eta) \\
N_7 &= \frac{1}{2}(1-\xi^2)(1+\eta) \\
N_8 &= \frac{1}{2}\,(1-\xi^2)(1-\eta^2)
\end{aligned} \tag{5.29}$$

and (ξ, η) are the same natural coordinates used in previous section 5.2 and shown in figure 5.4.

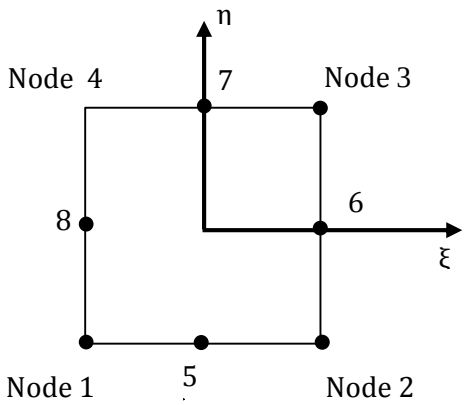

Figure 5.4 - Natural coordinates of the DKQ Element and its mid-nodes.

(2) The Kirchhoff hypothesis is imposed at:

(a) The corner nodes

$$\gamma = \begin{bmatrix} \beta_x + w_{,x} \\ \beta_y + w_{,y} \end{bmatrix} = 0 \qquad at\ nodes\ 1,2,3\ \ and\ 4 \tag{5.30}$$

(b) The mid-nodes

$$\beta_s + w_{,s_k} \quad k = 5,6,7\ and\ 8 \tag{5.31}$$

where s_k represents the directions aligned with the element sides.

(3) The variation of w along the sides is quadratic, i.e.

$$w_{,sk} = \frac{-3}{2l_{ij}}(w_i - w_j) - \frac{1}{4}(w_{,si} + w_{,sj}) \tag{5.32}$$

where,

k = *5,6,7,8* for *ij = 1-2, 2-3, 3-4, 4-1* and l_{ij} = length of the line connecting nodes *ij*

(4) A linear variation of β_{nk} is imposed along the side, i.e.

$$\beta_{nk} = \frac{1}{2}(\beta_{ni} \quad + \quad \beta_{nj}) = -\frac{1}{2}(w_{,ni} \quad + \quad w_{,nj}) \tag{5.33}$$

The β_x and β_y are obtained in terms of the displacements,

$$\beta_x = [H_x^T(\xi,\eta)]\{U\}$$

$$\beta_y = \left[H_y^T(\xi,\eta)\right]\{U\} \tag{5.34}$$

where $\{U\}$ are displacements at the nodes related to bending,

$$\{U\}^T = \{w_1 \quad \theta_{x_1} \quad \theta_{y_1} \quad w_2 \quad \theta_{x_2} \quad \theta_{y_2} \quad w_3 \quad \theta_{x_3} \quad \theta_{y_3} \quad w_4 \quad \theta_{x_4} \quad \theta_{y_4}\}$$

and the H_x and H_y , are the twelve component vectors of new shape functions,

$$[H^x(\xi,\eta)] = \begin{bmatrix} \frac{3}{2}(a_5 N_5 - a_8 N_8) \\ b_5 N_5 + b_8 N_8 \\ N_1 - c_5 N_5 - c_8 N_8 \\ \frac{3}{2}(a_6 N_6 - a_5 N_5) \\ b_6 N_6 + b_5 N_5 \\ N_2 - c_6 N_6 - c_5 N_5 \\ \frac{3}{2}(a_7 N_7 - a_6 N_6) \\ b_7 N_7 + b_6 N_6 \\ N_3 - c_7 N_7 - c_6 N_6 \\ \frac{3}{2}(a_8 N_8 - a_7 N_7) \\ b_8 N_8 + b_7 N_7 \\ N_4 - c_8 N_8 - c_7 N_7 \end{bmatrix} \tag{5.35}$$

and the H_y component vectors are,

$$[H^y(\xi,\eta)] = \begin{bmatrix} \frac{3}{2}(d_5 N_5 - d_8 N_8) \\ -N_1 - e_5 N_5 - e_8 N_8 \\ -b_5 N_5 - b_8 N_8 \\ \frac{3}{2}(d_6 N_6 - d_5 N_5) \\ -N_2 - e_6 N_6 - e_5 N_5 \\ -b_6 N_6 - b_5 N_5 \\ \frac{3}{2}(d_7 N_7 - d_6 N_6) \\ -N_3 - e_7 N_7 - e_6 N_6 \\ -b_7 N_7 - b_6 N_6 \\ \frac{3}{2}(d_8 N_8 - d_7 N_7) \\ -N_4 - e_8 N_8 - e_7 N_7 \\ -b_8 N_8 - b_7 N_7 \end{bmatrix} \tag{5.36}$$

where,

$$a_k = -\frac{x_{ij}}{l_{ij}^2}$$

$$b_k = \frac{3}{4}\frac{x_{ij} y_{ij}}{l_{ij}^2}$$

$$c_k = \left(\frac{1}{4}x_{ij}^2 - \frac{1}{2}y_{ij}^2\right)/l_{ij}^2$$

$$d_k = -\frac{y_{ij}^2}{l_{ij}^2} \tag{5.37}$$

$$e_k = \left(\frac{1}{4}y_{ij}^2 - \frac{1}{2}x_{ij}^2\right)/l_{ij}^2$$

$$x_{ij} = x_i - x_j$$

$$y_{ij} = y_i - y_j$$

$$l_{ij}^2 = \left(x_{ij}^2 + y_{ij}^2\right)$$

and,

$k = 5, 6, 7, 8$ for ij = 1-2, 2-3, 3-4, 4-1.

The strain-displacement matrix $[B]$ is obtained from the component vectors of the shape functions as,

$$[B(\xi,\eta)\,] = \begin{bmatrix} [H^x{}_{,x}] \\ [H^y{}_{,y}] \\ [H^x{}_{,y} + H^y{}_{,x}] \end{bmatrix} = \begin{bmatrix} j_{11}[H^x{}_{,\xi}] + j_{12}[H^x{}_{,\eta}] \\ j_{21}[H^y{}_{,\xi}] + j_{22}[H^y{}_{,\eta}] \\ j_{11}[H^x{}_{,\xi}] + j_{12}[H^x{}_{,\eta}] + j_{21}[H^y{}_{,\xi}] + j_{22}[H^y{}_{,\eta}] \end{bmatrix} \tag{5.38}$$

where the Jacobian is,

$$\lfloor J \rfloor = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} = \frac{1}{4}\begin{bmatrix} x_{21} + x_{34} + \eta(x_{12} + x_{34}) & y_{21} + y_{34} + \eta(y_{12} + y_{34}) \\ x_{32} + x_{41} + \xi(x_{12} + x_{34}) & y_{32} + y_{41} + \xi(y_{12} + y_{34}) \end{bmatrix} \tag{5.39}$$

and,

$$j_{11} = \frac{1}{det[J]}J_{22}$$

$$j_{12} = \frac{-1}{det\lfloor J \rfloor}J_{12} \tag{5.40}$$

$$j_{21} = \frac{-1}{det[J]}J_{21}$$

$$j_{22} = \frac{1}{det[J]}J_{22}$$

the determinant of the Jacobian is,

$$det[J] = \frac{1}{8}(y_{42}x_{31} - y_{31}x_{42}) + \frac{\xi}{8}(y_{34}x_{21} - y_{21}x_{34}) + \frac{\eta}{8}(y_{41}x_{32} - y_{32}x_{41}) \tag{5.41}$$

The derivatives $\left[H^{x}{}_{,\xi}\right]$, $\left[H^{x}{}_{,\eta}\right]$, $\left[H^{y}{}_{,\xi}\right]$, and $\left[H^{y}{}_{,\eta}\right]$ can be obtained by substituting the derivatives of the shape functions $N_{i,\xi}$ and $N_{i\eta}$ respectively in place of the shape functions N_i.

$$\left[N_{,\xi}\right] = \begin{bmatrix} \frac{1}{4}(2\xi+\eta)(1-\eta) \\ \frac{1}{4}(2\xi-\eta)(1-\eta) \\ \frac{1}{4}(2\xi+\eta)(1+\eta) \\ \frac{1}{4}(2\xi-\eta)(1+\eta) \\ -\xi\,(1-\eta) \\ \frac{1}{2}\,(1-\eta^2) \\ -\xi(1+\eta) \\ -\frac{1}{2}(1-\eta^2) \end{bmatrix} \tag{5.42}$$

$$\left[N_{,\eta}\right] = \begin{bmatrix} \frac{1}{4}(2\eta+\xi)(1-\xi) \\ \frac{1}{4}(2\eta-\xi)(1+\xi) \\ \frac{1}{4}(2\eta+\xi)(1+\xi) \\ \frac{1}{4}(2\eta-\xi)(1-\xi) \\ -\frac{1}{2}(1-\xi^2) \\ -\eta(1+\xi) \\ \frac{1}{2}(1-\xi^2) \\ -\eta(1-\xi) \end{bmatrix} \tag{5.43}$$

The stiffness matrix of the DKT element is obtained by,

$$[K] = \int_{-1}^{1}\int_{-1}^{1} [B]^T\,[D_b][B]det[J]\;d\xi d\eta \tag{5.44}$$

According to Batoz and Tahar [3], equation (5.44) can be solved by using a two point numerical integration scheme.

$$[K] = \sum_{j=1}^{2}\sum_{i=1}^{2} w_i w_j\,[B]^T[D_b][B]det[J] \tag{5.45}$$

The coordinates and the weight functions for this integration scheme are as following,

Table 5.1 - Coordinates and weight functions for Gauss quadrature.

Integration point	Coordinate	Weight functions
1	+0.577350269189626	1.0
2	–0.577350269189626	1.0

5.4. Quadrilateral Flat Shell Element Stiffness Matrix Formulation

The procedure to obtain the stiffness matrix of the quadrilateral flat shell element is the same used for the matrix of the triangular flat shell element. The only difference is in the dimension of the matrices of the elements from which the assembly is made. The stiffness matrix of the quadrilateral flat shell element at each node can then be represented by:

$$\left[K_{qfs}\right]_i = \begin{bmatrix} [K_m]_{2\times2} & [0]_{2\times3} & 0 \\ [0]_{3\times2} & [K_b]_{3\times3} & 0 \\ 0 & 0 & f \end{bmatrix} \tag{5.46}$$

where,

$\left[K_{qfs}\right]_i$ = Stiffness matrix at each node of the quadrilateral flat shell element.

$[K_m]_{2\times2}$ = Q4 stiffness matrix at each node of the shell element.

$[K_b]_{3\times3}$ = DKQ stiffness matrix at each node of the shell element.

f = Fictitious stiffness value of max $\left(\frac{(Kqfs)_{i,i}}{1000}\right)$.

5.5. Coordinate Transformation

The process for the coordinate transformation from local to global axis is the same as seen in the previous chapter, so it will not be repeated. The difference lays on the orientation of the local coordinate system, as shown in figure 5.5.

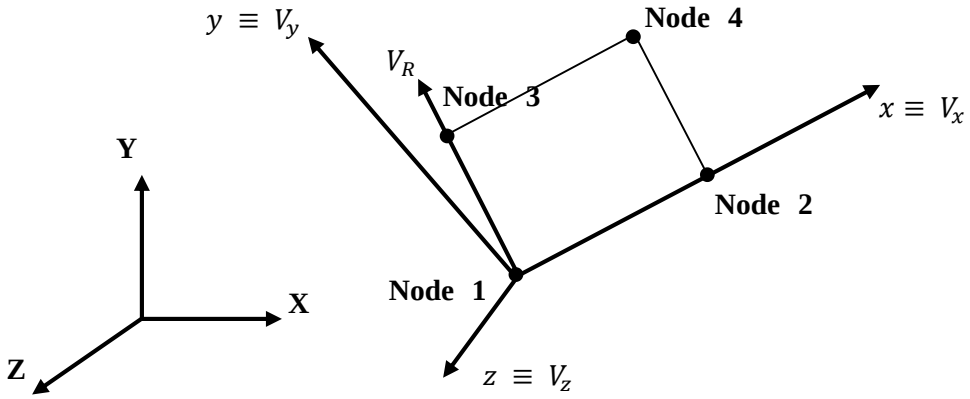

Figure 5.5 - Orientation of the local coordinate system.

The 3×3 transformation matrix is constructed with the cosine directios as for the triangular element.

$$[\lambda]_{3\times3} = \begin{bmatrix}\{\lambda_x\} & \{\lambda_y\} & \{\lambda_z\}\end{bmatrix} = \begin{bmatrix}(\lambda_x)_1 & (\lambda_y)_1 & (\lambda_z)_1 \\ (\lambda_x)_2 & (\lambda_y)_2 & (\lambda_z)_2 \\ (\lambda_x)_3 & (\lambda_y)_3 & (\lambda_z)_3\end{bmatrix} \tag{5.47}$$

and,

$$[T_p]_{6\times6} = \begin{bmatrix}[\lambda]_{3\times3} & [0]_{3\times3} \\ [0]_{3\times3} & [\lambda]_{3\times3}\end{bmatrix} \tag{5.48}$$

The transformation matrix can be obtained by repeating matrix (5.48),

$$[T] = \begin{bmatrix}[T_p] & [0] & [0] & [0] \\ [0] & [T_p] & [0] & [0] \\ [0] & [0] & [T_p] & [0] \\ [0] & [0] & [0] & [T_p]\end{bmatrix} \tag{5.49}$$

Chapter 6

6. Nonlinear Analysis and Corotational Formulation

6.1. Overview

This chapter is intended to study structural nonlinear problems, the formulations and the methods used to solve them. There are mainly 2 types of nonlinearity, which can be combined:

(1) Material – When the stress-strain relation characterizing the material is nonlinear, either because the material has elastic nonlinear behavior or exhibits plastic behavior. In this case strains can no longer be considered infinitesimal, and must be assumed finite. An example of material with nonlinear stress-strain is shown in figure 6.1.

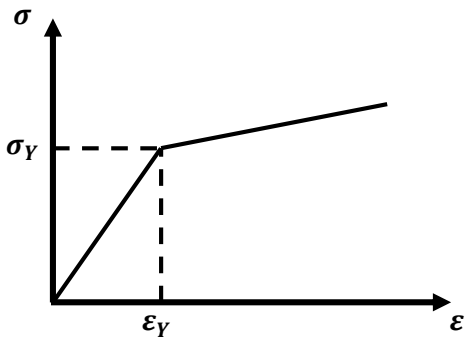

Figure 6.1 – Nonlinear stress-strain relation.

(2) Geometrical – When the displacements caused by the loading process are large enough to change the orientation of the internal forces and moments, the equilibrium equations must be written using the deformed geometry and that causes nonlinearities. It becomes necessary to distinguish between the initial and the deformed configurations. This type of nonlinearity is characterized by small strains and large displacements and rotations.

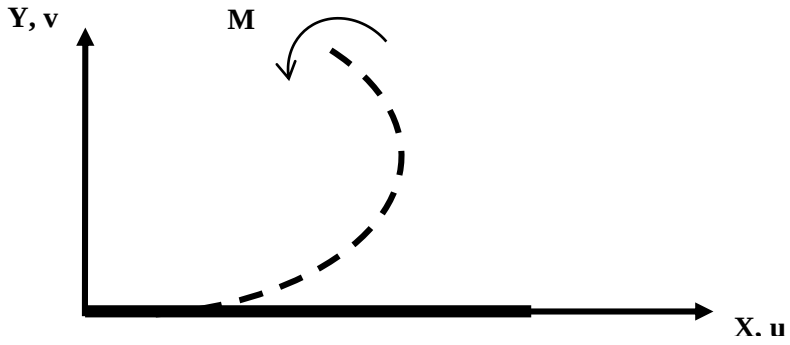

Figure 6.2 - Nonlinear geometric problem (large displacements and rotations).

The field of nonlinear finite element analysis is vast and complex, even if restricted to the cases of geometrical nonlinearity. The capability to solve geometric nonlinear problems involving large displacements and rotations was implemented in EFFECT, but restricted to models created with Beam2D elements. It should be noticed that all the necessary tools for

executing the nonlinear analysis were implemented, simplifying future developments. The metods employed to solve these problems involves the combination of the Newton-Raphson iterative method with a kinematic formulation. There are 3 most used kinematic formulations for this kind of problems:

(1) Total Lagrangian – The motion of the body is referenced to a fixed configuration, usually to the initial undeformed configuration.

(2) Updated Lagrangian – The motion of the body if referred to the latest known configuration.

(3) Corotational – Two reference configurations are defined; the base or the undeformed configuration, used to measure the displacements and the corotated configuration obtained through a rigid body motion, and used to measure stresses and deformations.

EFFECT uses the corotational kinematic formulation. The main reason why this formulation was chosed lies in its main advantage, the possibility of reusing the linear stiffness matrixes for the elements, derived using the conventional small-strain theory in the local system. The main disadvantage of the corotational formulation is the assumption that the displacements and rotations may be arbitrarily large but the deformations must be small, According to Felippa and Haugan [28], this is preventing its generalization on FEM codes. The concept of corotational formulation is based on the decomposition of the motion into two components:

(1) Base configuration which is kept fixed through all the analysis and is used to measure the displacements of the rigid body.

(2) Corotated configuration which is specific to each element and moves with it during the analysis. It is attached to and rotates with each element so that, when measuring displacements with respect to this corotated frame, the rigid body movement can be eliminated from the total displacements (measured through the base configuration). The deformations are assumed small, is then possible to use the small-strain theory to obtain the strains and element internal forces.

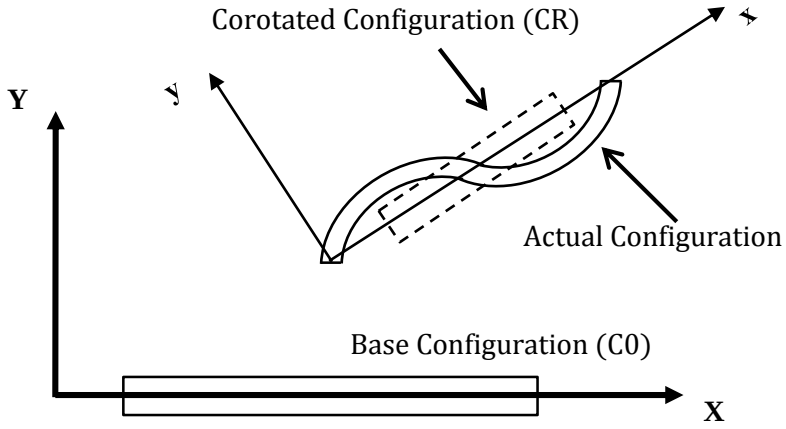

Figure 6.3 – The different configurations of the Corotational formulation.

6.2. Newton-Raphson Method

In linear problems, the structural equilibrium equations take the form,

$$[K]\{U\} = \{R\} \tag{6.1}$$

where $[K]$ and $\{U\}$ are independent from each other. However, when $[K]$ or $\{R\}$ or even both do become dependent on the displacements $\{U\}$ the problem expressed by (6.1) becomes nonlinear. In these cases it is not possible to obtain $\{U\}$ directly, it is necessary to make small-step increments from the last known equilibrium solution until the convergence criteria are met and the final equilibrium position is found.

The iterative method employed to solve the nonlinear finite element equations is the Newton-Raphson method, one of the most used procedures to solve nonlinear problems. The method is presented according to Cook [9]:

In the first iteration we have $\{u\} = \{0\}$ so the initial tangent stiffness matrix $[K_{t0}]$ is calculated regarding the undeformed configuration and the initial load increment is the load itself $\{\Delta P_1\} = \{P_F\}$ or the first load increment, $\{\Delta P_1\} = \{P_1\}$, if an incremental-iterative solution is sought. Solving (6.1), the first displacement increment is then,

$$\{\Delta u\} = [K_{t0}]^{-1}\{\Delta P_1\} \tag{6.2}$$

we can now obtain the first estimate of the final displacement $\{u_a\}$ by updating the solution,

$$\{u_a\} = \{0\} + \{\Delta u\} \tag{6.3}$$

Because the problem is nonlinear, the internal forces associated with $\{u_a\}$ do not equilibrate de applied loads and the load imbalance can be given by,

$$\{e_{Pa}\} = \{P_1\} - [K_{ta}]\{u_a\} \quad \text{where } K_{ta} \text{ is evaluated using the displacement } \{u_a\} \tag{6.4}$$

$[K_{ta}]\{u_a\}$ represents the sum of the internal forces on the structure in its current state of deformation. The intention is to reduce the load imbalance to zero. In the second iteration, a new displacement increment is calculated,

$$\{\Delta u\} = [K_{ta}]^{-1}\{e_{Pa}\} \tag{6.5}$$

the solution is once again updated to a more accurate displacement $\{u_b\}$,

$$\{u_b\} = \{u_a\} + \{\Delta U\} \tag{6.6}$$

and the load imbalance is recalculated,

$$\{e_{Pb}\} = \{P_1\} - [K_{tb}]\{u_b\} \quad \text{where } [K_{tb}] \text{ is evaluated using the displacement } \{u_b\} \tag{6.7}$$

These steps are repeated until the solution converges to the correct displacement $\{u_I\}$ corresponding to the load level, $\{P_1\}$ and the load imbalance is reduced to a sufficiently small value.

It is commom, when solving structural nonlinear problems, to adopt an incremental-iteractive approach. The total applied load $\{P_F\}$ is divided into a number of increments, $\{P_1\}$,

$\{P_2\}$, … $\{P_F\}$. Then the Newton-Rapson procedure described is used to converge to the equilibrium position for the first load level, $\{u_I\}$. When this is accomplished, the next load level, $\{P_2\}$ is considered and the Newton-Rapson procedure applied again in order to converge to the equilibrium position u_{II}, by computing the displacement increment,

$$\{\Delta u\} = [K_{tI}]^{-1}\{\Delta P_2\} \tag{6.8}$$

updating the solution,

$$\{u_c\} = \{u_I\} + \{\Delta u\} \tag{6.9}$$

and recalculating the load imbalance,

$$\{e_{PC}\} = \{P_2\} - [K_{tc}]\{u_c\} \quad \text{where } [K_{tc}] \text{ is evaluated using the displacement } \{u_c\} \tag{6.10}$$

If the imposed convergence criterion is still not satisfied, other displacement increments are computed, until convergence to level $\{P_2\}$. Then, the next load level is considered and this process is repeated until the equilibrium under the final load level is found. Figure 6.4, represents this approach.

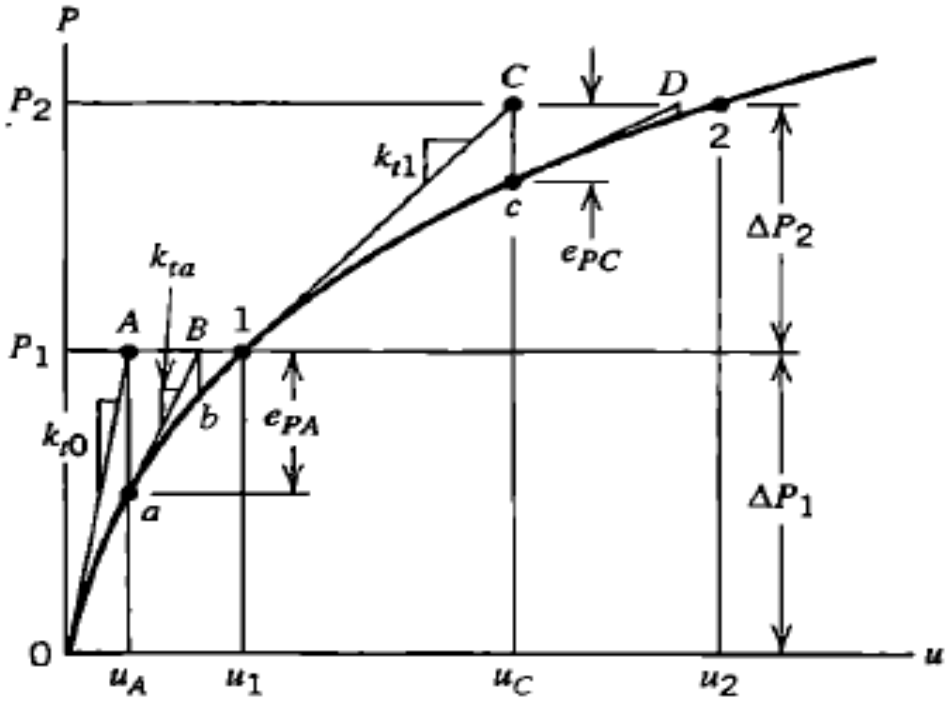

Figure 6.4 – Iteration to converge at each if load levels $\boldsymbol{P_1}$ **and** $\boldsymbol{P_2}$. [9].

The Newton-Raphson method has a very fast convergence rate, but according to Zienkiewicz *et al.* [11], is not without some negative features:

(1) A new tangent stiffness matrix has to be calculated in each iteration;

(2) Since direct solution is used for equation (6.2) the matrix needs to be factored at each iteration;

(3) On some occasions the tangent matrix is symmetric at a solution state but unsymmetrical otherwise. In these cases an unsymmetric solver is needed in general.

Although alternative procedures do not have some of these problems, the Newton-Raphson method has a quadratic asymptotic rate of convergence, which is not frequently seen in those other alternatives.

6.3. Corotational Formulation

The Corotational formulation implemented in EFFECT is based on the work of Ranking and Brogan [27]. Although in this dissertation only the Beam2D has been considered, the formulation is compatible with all the other elements.

To begin, Ranking and Brogan [27] assert that any displacement field can be decomposed into a rigid body motion and a strain-producing displacement.

$$\{u^{tot}\} = \{u^{def}\} + \{u^{rig}\} \tag{6.10}$$

where,

$\{u^{def}\}$ = displacements causing deformation

$\{u^{rig}\}$ = displacements associated with the rigid body motion.

The rigid body motion displacements, u^{rig}, can also be decomposed into a rigid body rotation and a rigid body translation. The translations do not produce any strain, according to the finite element formulation used, and do not need particular atention. So the corotational formulation only requires extracting the displacements associated with the rigid body rotations from the total displacements. The goal is to obtain u^{def}, the displacements causing deformation. For a Beam2D element, these can be decomposed into two translational and one rotational component.

6.3.1. Translational Components of Displacements Causing Deformation

To obtain the translational component of $\{u^{def}\}$ it is necessary to extract the displacements associated with the rigid body motion $\{u^{rig}\}$. To do so, a set of element coordinates is defined which follow the element as it deforms. The relationship between the global and the element or corotated coordinates can be expressed by,

$$\{X_g\} = [T_k]\{X_e\} \tag{6.11}$$

where,

$\{X_g\}$ = Element coordinates on the global system.

$\{X_e\}$ = Element coordinates on the local system.

$[T_k]$ = Transformation matrix.

The subscripts *k* refers to an iteration step, in this case, of the Newton-Raphson Method. The undeformed position after the rigid body rotation is given by,

$$\{X'_g\} = [T_k][T_0]^T\{X_g\} \tag{6.12}$$

where,

$\{X'_g\}$ = undeformed position expressed in the rotated corotational frame.

$[T_0]^T$ = Transformation matrix relating local and corotated frames.

since,

$$\{u_g^{rig}\} = \{X'_g\} - \{X_g\} \tag{6.13}$$

combining with the equation (6.12),

$$\{u_g^{rig}\} = [T_k][T_0]^T\{X_g\} - \{X_g\} = ([T_k][T_0]^T - [I])\{X_g\} \tag{6.14}$$

if the equation (6.10) is given in terms of the global coordinate system,

$$\{u_g^{tot}\} = \{u_g^{rig}\} + \{u_g^{def}\}$$

then with equation (6.14),

$$\{u_g^{def}\} = \{u_g^{tot}\} - ([T_k][T_0]^T - [I])\{X_g\} \tag{6.15}$$

into the element coordinate system,

$$\{u_e^{def}\} = [T_k]^T(\{u_g\} + \{X_g\}) - \{X_e\}) \tag{6.16}$$

In figure 6.5 it is possible to get the visual idea of this procedure. The rotational component of the deformations was ignored to perceive only the translational $\{u_e^{def}\}$. In figure 6.5, the initial length of the element is given by L_0 and at the actual configuration the length is L_n.

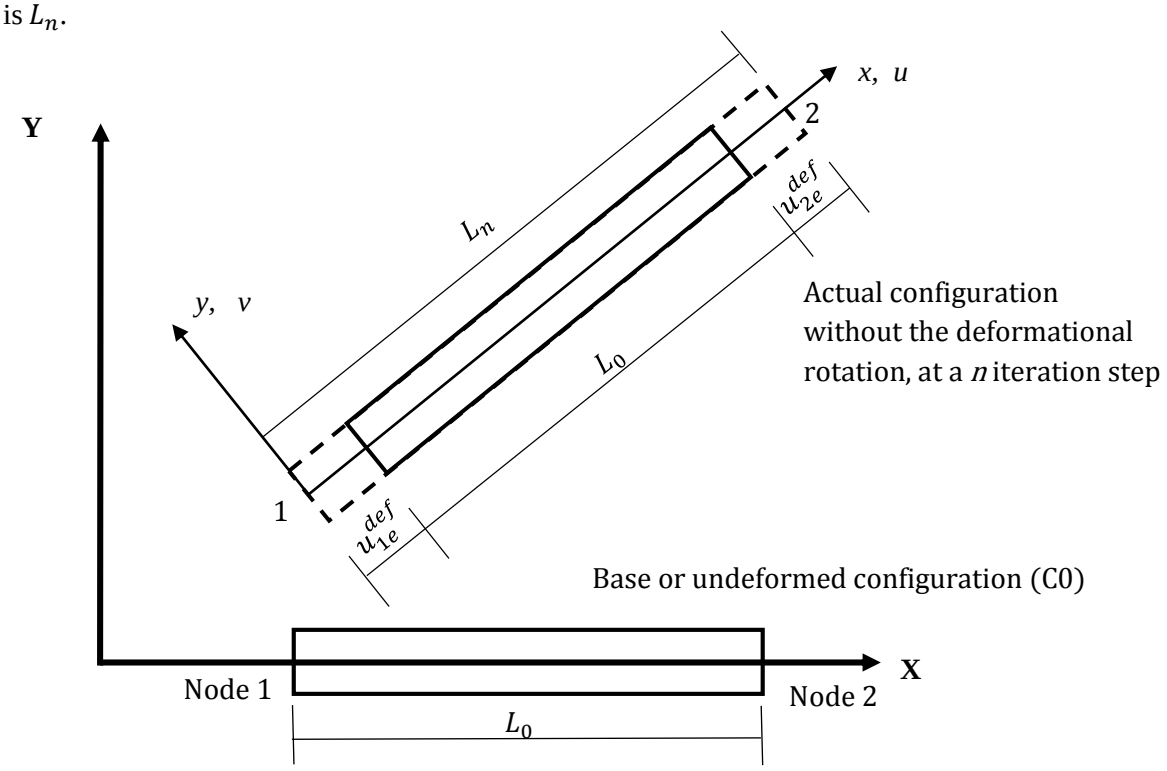

Figure 6.5 - Representation of the base configuration and the actual configuration at the *n* iteration step.

6.3.2. Rotational Component of the Displacements Causing Deformation

For the **Beam2D** element, the rotation component $\{\theta^{def}\}$ can be obtained by subtracting the rigid body rotations $\{\theta^{rig}\}$ directly from the total rotations $\{\theta^{tot}\}$:

$$\{\theta^{def}\} = \{\theta^{tot}\} - \{\theta^{rig}\} \Leftrightarrow \tag{6.17}$$

$$\Leftrightarrow \begin{Bmatrix} \theta_1^{def} \\ \theta_2^{def} \end{Bmatrix} = \begin{Bmatrix} \theta_1^{tot} \\ \theta_2^{tot} \end{Bmatrix} - \begin{Bmatrix} \theta^{rig} \\ \theta^{rig} \end{Bmatrix}$$

The total rotations $\{\theta^{tot}\}$ are taken directly from the global displacements obtained in each iteration by solving the nonlinear system of equations. Before describing the procedure to obtain $\{\theta^{rig}\}$, the figure 6.6 shows the different angles represented in equation (6.17).

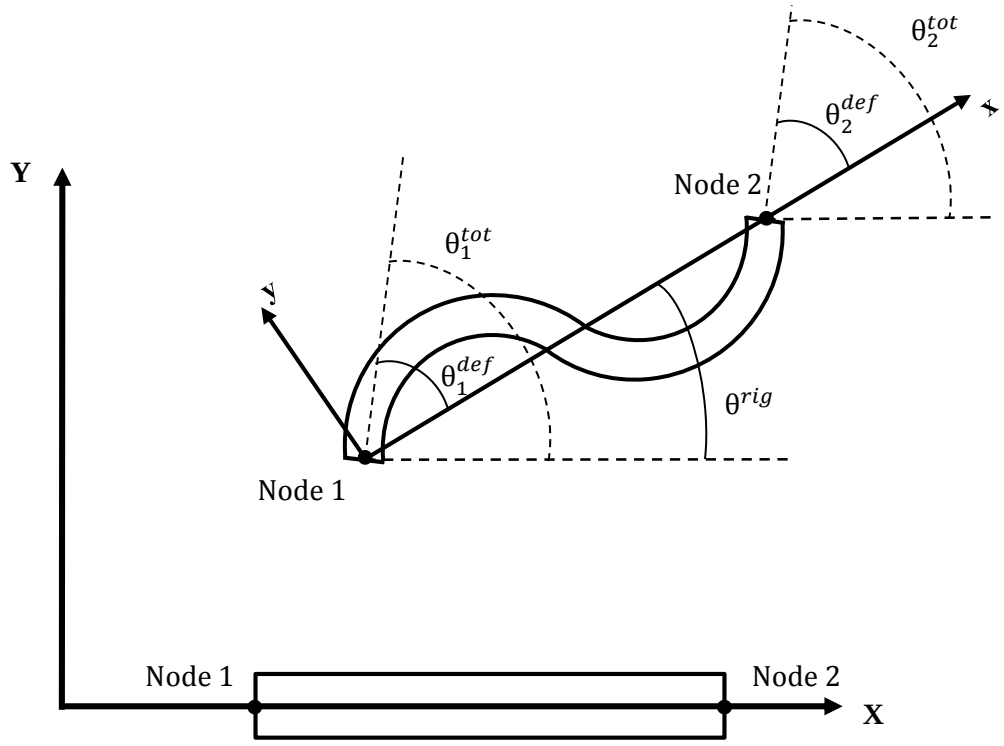

Figure 6.6 - Representation of the different rotations associated with the Corotational formulation.

To find the rigid body rotation of the element, the procedure starts by calculating the angle between the global coordinate system and the corotated coordinate system. The only inconvenience about this procedure is the possibility of multiple results, after all $\theta^{rig} = \theta^{rig} - 2\pi$, as it is shown below.

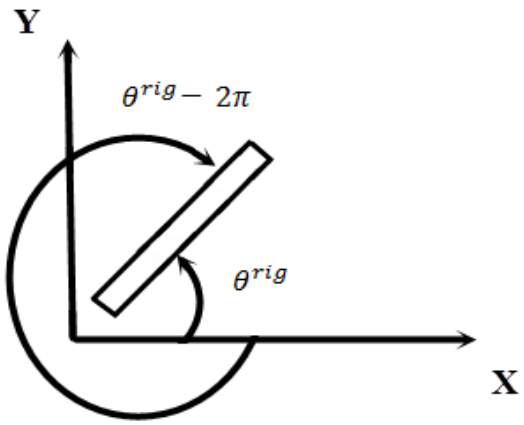

Figure 6.7- Angle inconvenience.

This inconvenience is dealt by first defining,

$$\theta^{rig} = \theta^{rig} + 2\pi k \qquad k \in [-3, +3] \tag{6.18}$$

by finding the correct k, the direction and number of rotations can be determined. The procedure developed to find k compares the average between the rotations of node 1 and 2 obtained from the previous iteration,

$$\theta_{average} = \frac{\theta_1^{tot} + \theta_2^{tot}}{2} \tag{6.19}$$

with all the possible values of $\theta^{rig} + 2\pi k$ between the range defined in (6.18); the *k* for which the two value of the angles are closer is the correct one. In other words the correct *k* is the one, who minimizes *d* in equation (6.20),

$$d = \theta^{rig} + 2\pi k - \theta_{average} \tag{6.20}$$

With θ^{rig} calculated it is possible to obtain T_0^T and compute u_e^{def} with equation (6.16) and also θ^{def} with equation (6.17).

6.3.3. Internal Forces

In a geometric nonlinear problem the internal forces are function of the displacement field. The internal forces in the Beam2D element are the axial force *N*, the shear force V_s and the bending moment at the nodes. If no distributed loads are applied along the element, the axial force and the shear force are constant, though the bending moments are not. Then M_1 and M_2 represent respectively the bending moments at node 1 and 2. According to Harrison [47] these forces can be calculated using the following equations,

$$N = EA_0\varepsilon = \frac{EA_0}{L_0}d \qquad V_s = \frac{M_1 + M_2}{L} = \frac{6EI}{LL_0}\left(\theta_1^{def} + \theta_2^{def}\right)$$

$$M_1 = \frac{2EI}{L_0}\left(2\theta_1^{def} + \theta_2^{def}\right) \qquad M_2 = \frac{2EI}{L_0}\left(\theta_1^{def} + 2\theta_2^{def}\right) \tag{6.21}$$

where,

E= Young's Modulus;

I = Inertia of the element.

L_0 is the initial length of the element at its undeformed configuration; d is the difference between L_0 and the length of the actual configuration, or the sum of u_e^{def} from both nodes, these relationships can be seen in figure 6. 8.

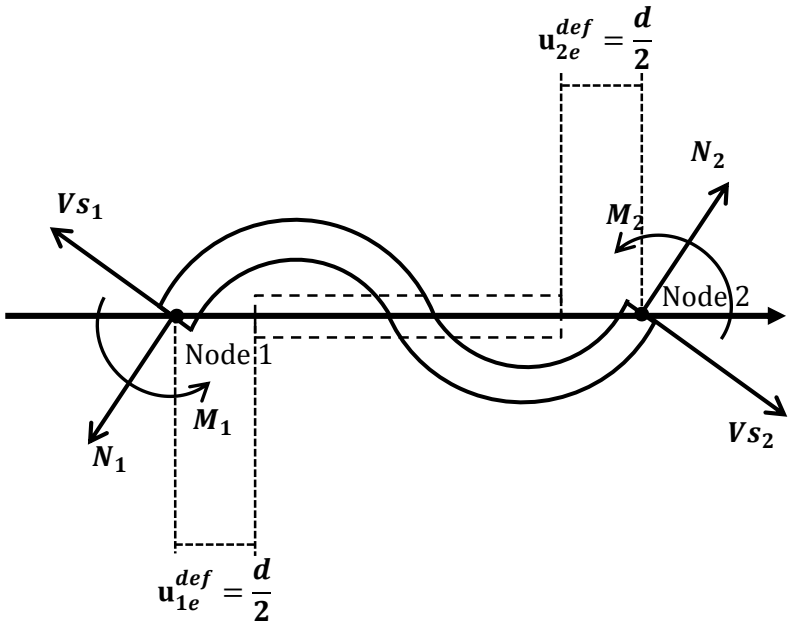

Figure 6.8 - Internal forces in a **Beam2D** element.

6.3.3. Tangent Stiffness Matrix Formulation

In the corotational formulation it is necessary to compute the so-called tangent matrix to deal with the geometric nonlinearity. This matrix represents the stiffness of the structure in the deformed configuration. One way to obtain this tangent matrix is given by the integral (6.22) using the usual strain-displacement relationship,

$$[K_t] = \int_{vol} [T_n]^T [B]^T [D][B][T_n]\, dV \tag{6.22}$$

Where,

$[B]$ = usual strain-displacement matrix;

$[T_n]$ = orthogonal transformation relating the original element coordinates to the corotated.

However, in the implementation of Beam2D in EFFECT, the exact nonlinear strain-displacement matrix was not considered. Instead, a simplified approach was taken.

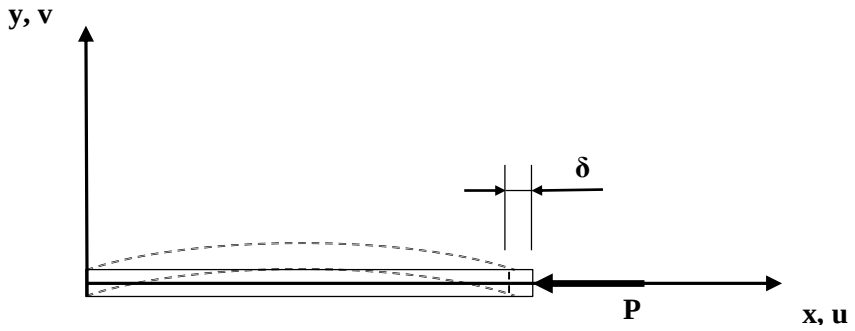

Figure 6.9 - Nonlinear geometric problem.

For a Beam2D represented in fig. 6.9, energy method can be applied to formulate the tangent stiffness matrix with some simplifications; the procedure to obtain this tangent stiffness matrix begins with the formulation of the geometric matrix:

The displacement δ from figure 6.9 can be approximated by,

$$\delta = \frac{1}{2}\int_0^L v_{,x}^2 \, dx \tag{6.23}$$

since the energy of load *P* is given by,

$$\mathrm{W} = -P\delta = \frac{P}{2}\int_0^L v_{,x}^2 \, dx \tag{6.24}$$

using the shape functions obtained in (3.23),

$$\mathrm{W} = -\frac{P}{2}\{u\}^T \int_0^L \left[N_{,x}\right]^T \left[N_{,x}\right] dx \, \{u\} \tag{6.25}$$

or,

$$\mathrm{W} = -\frac{P}{2}\{u\}^T [G]\{u\} \tag{6.26}$$

where ,

$$[G] = \int_0^L [N]_{,x}^T \, [N]_{,x} \, dx \tag{6.27}$$

is the geometric stiffness matrix, which represents the effects of the axial load *P* on the stiffness of the structure,

$$[G] = \begin{bmatrix} \frac{6}{5L} & \frac{1}{10} & -\frac{6}{5L} & \frac{1}{10} \\ \frac{1}{10} & \frac{2L}{15} & -\frac{1}{10} & -\frac{L}{30} \\ -\frac{6}{5L} & -\frac{1}{10} & \frac{6}{5L} & -\frac{1}{10} \\ \frac{1}{10} & -\frac{L}{30} & -\frac{1}{10} & \frac{2L}{15} \end{bmatrix} \tag{6.28}$$

Recalling that the Beam2D has 6 degrees of freedom, it is necessary to increase the dimension of the matrix (6.28) accordingly,

$$[G] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{6}{5L} & \frac{1}{10} & 0 & -\frac{6}{5L} & \frac{1}{10} \\ 0 & \frac{1}{10} & \frac{2L}{15} & 0 & -\frac{1}{10} & -\frac{L}{30} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{10} & \frac{1}{10} & 0 & \frac{6}{5L} & -\frac{1}{10} \\ 0 & \frac{1}{10} & \frac{1}{10} & 0 & -\frac{1}{10} & \frac{2L}{15} \end{bmatrix} \tag{6.29}$$

Finally the stiffness matrix, considering the effect of the axial forces is given by,

$$[K_t] = [K] + N[G]$$

where $[K]$ is the linear stiffness matrix and *N* is the axial internal force of the element. The tangent stiffness matrix is obtained assembling all the individual $[K_t]$, after applying the adequate coordinate transformation, related to the corotated coordinate system.

6.3.4. Nonlinear Algorithm Scheme

The schematic of the nonlinear algorithm obtained by combining the Newton-Raphson Method with the corotational formulation is as follows.

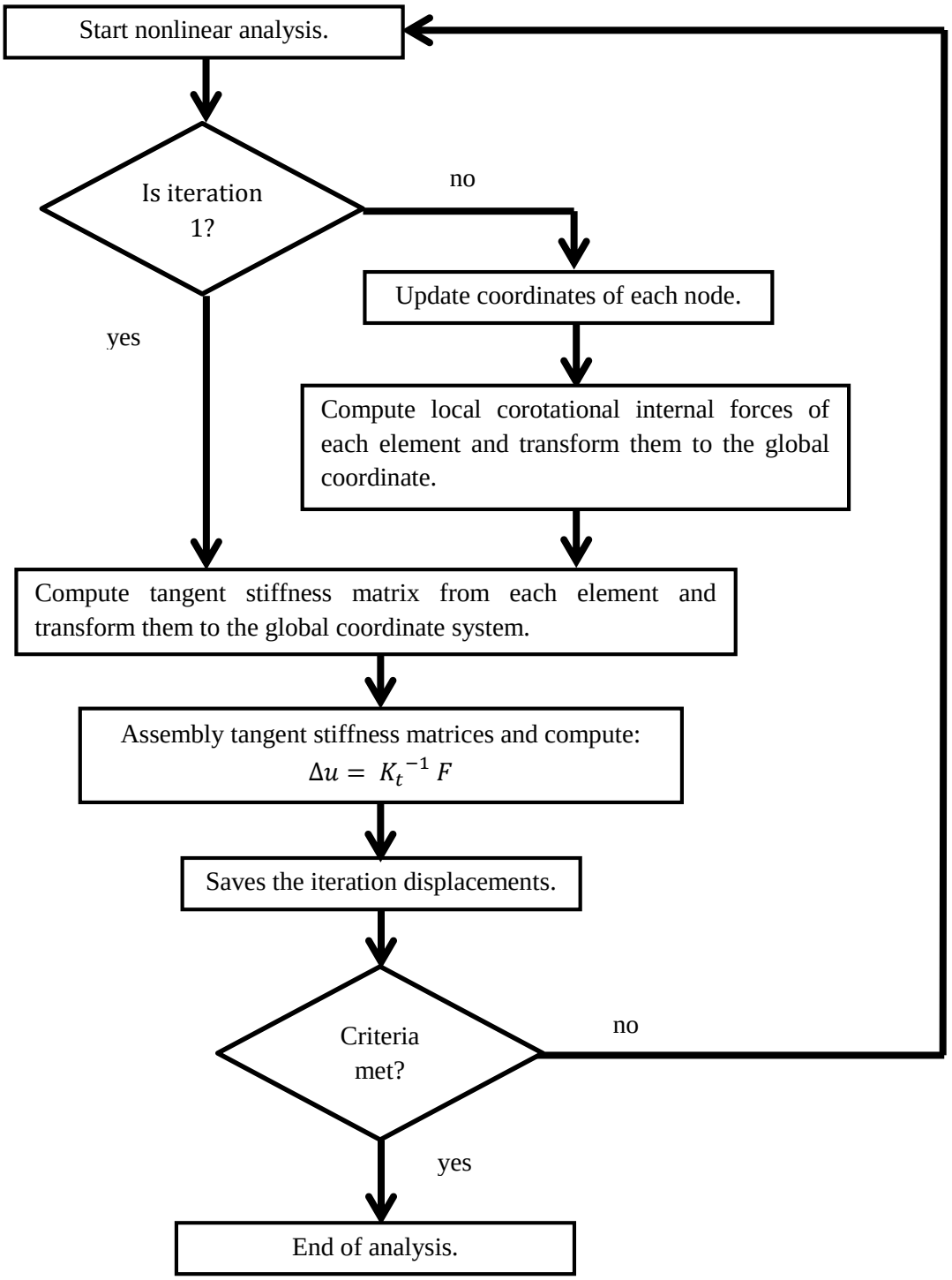

Figure 6.10 - Nonlinear analysis algorithm schematic

Chapter 7

7. Optimization and Sensitivity Analysis

7.1. Overview

This chapter presents the features implemented in EFFECT in order to solve structural optimization problems and the detailed description of the analytic expressions used in sensitivity analysis. EFFECT was designed to be incorporated into a powerful optimization tool, capable of solving a variety of optimization problems, includind Reliability-Based Design Optimization (RBDO) and Robust Design Optimization (RDO) [48]. To accomplish this objective, two important features where developed: a) interfaces allowing the information to be echanged with the optimization algorithm during the optimization iterative process and b) the capacity to calculate the derivatives of functions used in the formulation of optimization problems with respect to design variables by the continuum method of design sensitivity analysis.

The optimization system in which EFFECT is integrated was developed as part of a Msc. Thesis [49] and has a modular structure with 3 modules:

1) FEM program capable of computing sensitivities (EFFECT);
2) A structural reliability analysis program;
3) Optimization algorithm (SQP algorithm implemented in MATLAB toolbox).

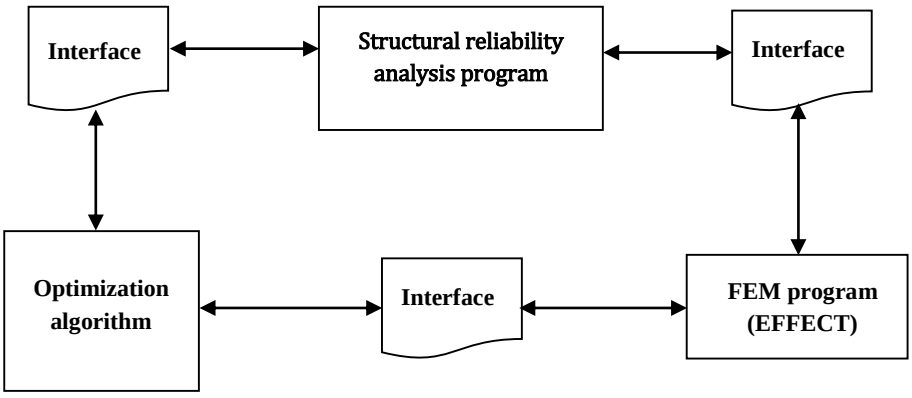

Figure 7.1 - Diagram showing the modules of the software platform.

As the interfaces between EFFECT and the optimization algorithm or the structural reliability analysis program are exactly the same, and as the full system presented in figure 7.1 is compreensively described in [49], only a simplified version of it, shown in figure 7.2, will be presented here. Consequently, all the details related with structural reliability analysis will not be covered in this thesis.

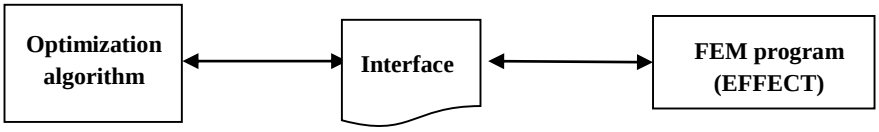

Figure 7.2 - Simpler version of the optimization scheme of the fig. 7.1.

7.2. Structural Optimization Formulation

The classical mathematical formulation of a continuous structural optimization problem can be stated as,

Obtain the value of the project variables $b = \{b_i\}$, $i = 1, \ldots, N$ which,

Minimizes F(b)

$$\text{Subject to,} \quad g_j\big(\Psi(b)\big) \leq 0 \qquad j = 1, \ldots, M \qquad (7.1)$$
$$h_k\big(\Psi(b)\big) = 0 \qquad k = 1, \ldots, L$$

$$\text{And the conditions } b_i^- \leq b_i \leq b_i^+ \qquad i = 1, \ldots, N$$

where F(b) is the objective function, $\Psi(b)$ are the functions that represent the structural performance, g_j and h_j represent the set of inequality and equality constraints, respectively. Finally b_i^- and b_i^+ are the lower and upper bounds of the value of the project variables.

This formulation is implemented in a MATLAB program and then solved by the SQP algorithm contained in its toolbox *optimtool*. The SQP is an iterative algorithm and requires the objective F(b) and performance functions $\Psi(b)$ gradients, which are calculated in EFFECT through the continuum method of design sensitivity analysis.

7.3. Shape Sensitivity Analysis

This section presents the methodology used for obtaining the analytic expressions applied in shape sensitivity computation. For sensitivity analysis it is meant the calculation of the gradients of performances with respect to design variables. The design variables can be classified into 4 categories:

(1) Materials – when they affect the value of material proprieties (e.g. Young's modulus);
(2) Load – when they affect the value of the applied load;
(3) Size – when they affect the size of the cross-section (e.g. Area of the cross-section).
(4) Shape – when they affect the shape of the structure (e.g. Nodal coordinates).

There are 2 different methods to obtain the analytic expressions for the sensitivity calculation, the discrete and the continuum method. In the discrete approach the derivatives are obtained by discretizing and then differentiating the structural equations with respect to the design variables, whereas in the continuum method, the partial differential equations, obtained from the principle of virtual work, are derived with respect to the design variables and the resulting equations are then discretized. In either case, the efficient solution of these equations requires one of the two possible approachs, the direct or the adjoint method. The most adequate choice depends on the number of performances and design variables of the problem. The direct approach needs to solve a linear system of equations for each design variable while the adjoint approach requires the solution of a similar system for each performance. As the number of performances is normally much smaller than the number of design variables, the adjoint approach was implemented in EFFECT into the Beam2D and Truss2D elements.

The methodology used was obtained from Cardoso [46] based on the work of Haug *et al.* [44]. The continuum method of design sensitivity analysis and the adjoint approach were used to obtain gradients of structural performances with respect to material, load and size design variables. The concept of material derivative was further used to take in account the shape variations. This concept and the sensitivity expressions are described in detail in the next sections of this chapter.

7.3.1. Material Derivate of a Functional

The concept of material derivate is used to obtain the analytic expressions for a sensitivity analysis involving shape variations. To understand the concept it is necessary to first define Ω and Ω_τ, which are two continuum domains. The transformation T between the two domains is defined only by the τ parameter, as it is illustrated in the figure 7.3.

$$x_\tau = T(x,\tau) \qquad (7.2)$$
$$\Omega_\tau \equiv T(\Omega,\tau)$$

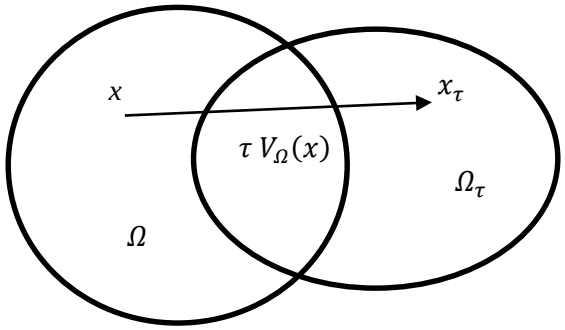

Figure 7.3 – Shape variation of Ω domain.

Initially $\tau = 0$ and the domain is Ω. During the transformation process the points that define the shape of the structure move to the $x_\tau = T(x,\tau)$ position, the velocity field associated to this shape variations is given by,

$$V(x_\tau,\tau) \equiv \frac{dx_\tau}{d\tau} = \frac{dT(x,\tau)}{d\tau} = \frac{\partial T(x,\tau)}{\partial \tau} \qquad (7.3)$$

The material derivative is defined in the $x \in \Omega$ point, if it exists, by,

$$\dot{z}(x) = \frac{dx_\tau}{d\tau} z_\tau\big(x + \tau V(x)\big)\Big|_{\tau=0} = \lim_{\tau\to 0}\left[\frac{z_\tau\big(x + \tau V(x)\big) - z(x)}{\tau}\right] \qquad (74)$$

If z_t has a regular extension to the neighborhood of Ω_t then,

$$\dot{z}(x) = z'_v(x) + \nabla z^T V(x) \qquad (7.5)$$

where,

$$z'_v(x) = \lim_{\tau\to 0} \frac{z_\tau(x) - z(x)}{\tau} \qquad (7.6)$$

is the partial derivative of z due to the shape variation V and $\nabla z = [z_{,1} \quad z_{,2} \quad z_{,3}]$ is the gradient of z regarding to X. The material derivate concept can also be applied to a functional,

$$\Psi = \int_{\Omega} f(x)\, d\Omega \tag{7.7}$$

obtaining,

$$\Psi' = \int_{\Omega} [f'_v(x) + \nabla f(x)^T V(x) + f(x) div V(x)]\, d\Omega \tag{7.8}$$

7.3.2. Sensitivities of Shape Variations

For a problem with boundary conditions the equilibrium equation can be written using the variational approach,

$$a(z, \bar{z}) \equiv \int_{\Omega} c(z, \bar{z}) d\Omega = \int_{\Omega} f^T \bar{z}\, d\Omega \equiv l(\bar{z}) \qquad to\ all\ \bar{z} \in Z \tag{7.9}$$

Where $c(\cdot,\cdot)$ is a bilinear function, associated with the elastic energy, f is the external concentrated loads, $l(\bar{z})$ is the term associated with the distributed loads and $\bar{z}$ are the kinetically admissible virtual displacements. If the $\dot{z}$ material derivative does exist, then the expression from (7.9) can be differentiable regarding the shape variation of the domain.

$$[a(z, \bar{z})]' = \int_{\Omega} [c(z'_v, \bar{z}) + c(z, \bar{z}'_v) + \nabla c(z, \bar{z})^T V + c(z, \bar{z}) div V] d\Omega \tag{7.10}$$

since,

$$\dot{z} = \bar{z}'_v + \nabla z^T V \tag{7.11}$$

one obtains,

$$[a(z, \bar{z})]' = \int_{\Omega} [c(\dot{z} - \nabla z^T V, \bar{z}) + c(z, \dot{\bar{z}} - \nabla \bar{z}^T V) + \nabla c(z, \bar{z})^T V + c(z, \bar{z}) div V] d\Omega \tag{7.12}$$

if,

$$\dot{\bar{z}} = \bar{z}'_v + \nabla \bar{z}^T V = 0 \tag{7.13}$$

then,

$$[a(z, \bar{z})]' = \int_{\Omega} [c(\dot{z}, \bar{z}) - c(\nabla \bar{z}^T V, \bar{z}) - c(z, \nabla \bar{z}^T V) + \nabla c(z, \bar{z})^T V + c(z, \bar{z}) div V] d\Omega \tag{7.14}$$

combining the second term of (7.9) with (7.8)

$$[l(\bar{z})]' = \int_{\Omega} [f_v^{T\prime} \bar{z} + f^T \bar{z}'_v + \nabla (f^T \bar{z})^T V + f^T \bar{z}\, div V] d\Omega \tag{7.15}$$

$$[l(\bar{z})]' = \int_{\Omega} [f_v^{T\prime} \bar{z} + f^T (\dot{\bar{z}} - \nabla \bar{z}^T V) + \nabla (f^T \bar{z})^T V + f^T \bar{z}\, div V] d\Omega \tag{7.16}$$

$$[l(\bar{z})]' = \int_{\Omega} [f_v^{T\prime}\bar{z} - f^T(\nabla\bar{z}^T V) + \nabla(f^T\bar{z})^T V + f^T\bar{z}\, divV]d\Omega \tag{7.17}$$

rewriting (7.9) with (7.12) and (7.15),

$$\int_{\Omega} [c(\dot{z},\bar{z}) - c(\nabla\bar{z}^T V,\bar{z}) - c(z,\nabla\bar{z}^T V) + \nabla c(z,\bar{z})^T V + c(z,\bar{z})divV]d\Omega$$
$$= \int_{\Omega} [f_v^{T\prime}\bar{z} - f^T(\nabla\bar{z}^T V) + \nabla(f^T\bar{z})^T V + f^T\bar{z}\, divV]d\Omega \tag{7.18}$$

or,

$$a(\dot{z},\bar{z}) = l'_v(\bar{z}) - a'_v(z,\bar{z}) \tag{7.19}$$

where,

$$a(\dot{z},\bar{z}) = \int_{\Omega} c(\dot{z},\bar{z})\, d\Omega \tag{7.20}$$

$$l'_v(\bar{z}) = \int_{\Omega} [f_v^{T\prime}\bar{z} - f^T(\nabla\bar{z}^T V) + \nabla(f^T\bar{z})^T V + f^T\bar{z}\, divV]d\Omega \tag{7.21}$$

$$a'_v(z,\bar{z}) = -\int_{\Omega} [-c(\dot{z},\bar{z}) + c(\nabla\bar{z}^T V,\bar{z}) + c(z,\nabla\bar{z}^T V) - \nabla c(z,\bar{z})^T V - c(z,\bar{z})divV]d\Omega \tag{7.22}$$

The equation (7.19) represents the material derivative of (7.9) for shape variations, expressed by the velocity field *V*, defined in (7.3).

Considering a generic functional,

$$\Psi = \int_{\Omega} g(z,\nabla z)\, d\Omega \tag{7.23}$$

According with (7.8) the material derivative of the functional is,

$$\Psi = \int_{\Omega} [g_z z'_v + g_{\nabla z}\nabla z'_v + \nabla g^T V + g\, divV]\, \mathrm{d}\Omega \tag{7.24}$$

with,

$$g_z = \frac{\partial g(z,\nabla\tilde{z})}{\partial z} \qquad and \qquad g_{\nabla z} = \frac{\partial g(\tilde{z},\nabla z)}{\partial(\nabla z)} \tag{7.25}$$

where $\tilde{z}$ is kept constant during the differentiation. Combining the expression from (7.20) with the equation (7.11) one obtains,

$$\Psi' = \int_{\Omega} [g_z\dot{z} + g_{\nabla z}\nabla\dot{z} - g_z(\nabla z^T V) - g_{\nabla z}\nabla(\nabla z^T V) + \nabla g^T V + g\, divV]\, d\Omega \tag{7.26}$$

To get equation (7.26) explicitly in function of the velocity field it is necessary to express the material derivative of the displacements in function of the velocity field associated with the shape variation, to accomplish this the adjoint method was used.

7.3.3. Adjoint Method

An adjoint equation is obtained by substituting $\dot{z} \in \mathrm{Z}$ by $\bar{\lambda} \in \mathrm{Z}$ in the equation (7.26) and matching the bilinear energetic terms of the equation (7.9) with the terms of (7.26) which contain the parameter $\bar{\lambda}$,

$$a(\lambda, \bar{\lambda}) = \int_{\Omega} \left[g_z \bar{\lambda} + g_{\nabla z} \nabla \bar{\lambda} \right] \mathrm{d}\Omega \quad for\ all\ \bar{\lambda} \in Z \tag{7.27}$$

The equation (7.27) has only one solution λ, which is called adjoint variable, associated with the performance Ψ.

$$a(\dot{z}, \lambda) = \int_{\Omega} \left[g_z \dot{z} + g_{\nabla z} \nabla \dot{z} \right] \mathrm{d}\Omega \tag{7.28}$$

replacing $\bar{\lambda}$ for $\dot{z}$ in the (7.19) gives,

$$a(\dot{z}, \lambda) = l'_v(\lambda) - a'_v(z, \lambda) \tag{7.29}$$

since the energetic term is symmetric,

$$a(\lambda, \bar{\lambda}) = a(\dot{z}, \lambda) \tag{7.30}$$

then combining the equations (7.28) and (7.29) gives,

$$\int_{\Omega} \left[g_z \dot{z} + g_{\nabla z} \nabla \dot{z} \right] d\Omega = l'_v(\lambda) - a'_v(z, \lambda) \tag{7.31}$$

using the adjoint method the equation (7.26) can finally be written in function of the velocity field *V* expressed in (7.3),

$$\Psi' = l'_v(\lambda) - a'_v(z, \lambda) - \int_{\Omega} \left[g_z (\nabla z^T V) + g_{\nabla z} \nabla (\nabla z^T V) - \nabla g^T V - g\ div V \right] d\Omega \tag{7.32}$$

The equation (7.32) can be calculated when the velocity field *V*, the displacements from the original structure, *z*, and the adoint displacements, λ, are known.

The $l'_v(\lambda)$ term presented in equation (7.21) represents the energy associated with the distributed loads, which are not considered in this dissertation. The $a'_v(z, \lambda)$, alpha prime, term presented in equation (7.22) represents the energy associated with the internal loads and is calculated in different ways for the different structural elements as is going to be shown in the next sections. For the next section it is assumed the coordinate frame system shown in figure 7.4.

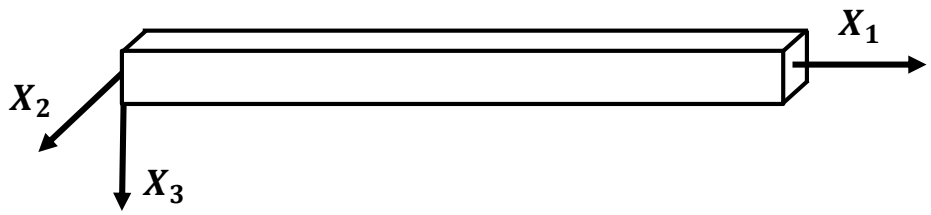

Figure 7.4 – Frame system for the sensitivities calculations.

7.3.4. Alpha Prime Calculation for the Truss2D Element

When a structural component deformes due to axial loads, the energy term of equation (7.9) becomes,

$$a(z,\bar{z}) \equiv \int_{\Omega} c(z,\bar{z})d\Omega = \int_{0}^{L} EAz_{1,1}\bar{z}_{1,1}\,dx_1 \tag{7.33}$$

Where *A* is the area of the cross-section and *E* is the Young's modulus. Because the elastic energy is expressed only in function of the axial displacements, only the first components of the vectors $\nabla z^T V$ and $\nabla \bar{z}^T V$, that is, $z_{1,1}V_1 + z_{1,2}V_2 + z_{1,3}V_3$ and $\bar{z}_{1,1}V_1 + \bar{z}_{1,2}V_2 + \bar{z}_{1,3}V_3$ are relevant. The four terms that make up the equation (7.22) are,

$$c(\nabla z^T V,\bar{z}) = EA\big(z_{1,11}V_1 + z_{1,1}V_{1,1} + z_{1,21}V_2 + z_{1,2}V_{2,1} + z_{1,31}V_3 + z_{1,3}V_{3,1}\big)\bar{z}_{1,1} \tag{7.34}$$

$$c(z,\nabla \bar{z}^T V) = EAz_{1,1}\big(\bar{z}_{1,1}V_1 + 2\bar{z}_{1,21}V_2 + \bar{z}_{1,2}V_{1,11} + \bar{z}_{3,211}V_2 + 2\bar{z}_{3,21}V_{2,1}\big)z_{3,11} \tag{7.35}$$

$$\begin{aligned}\nabla c(z,\bar{z})^T V = EA\big[&\big(z_{1,11}\bar{z}_{1,11} + z_{1,1}\bar{z}_{1,11}\big)V_1 + \big(z_{1,12}\bar{z}_{1,1} + z_{1,1}\bar{z}_{1,12}\big)V_2 \\ &+ \big(z_{1,13}\bar{z}_{1,1} + z_{1,1}\bar{z}_{1,13}\big)V_3\big]\end{aligned} \tag{7.36}$$

$$c(z,\bar{z})\,divV = EAz_{1,1}\bar{z}_{1,1}\big(V_{1,1} + V_{2,2} + V_{3,3}\big) \tag{7.37}$$

Since the order of derivation doesn't affect the value of the derivative $z_{1,1i} = z_{1,i1}$, the performance measures aren't affected by the translation of the structural component. By that reason V_2 and V_3 are only relevant when they concern the rotation of the truss, allowing $V_{2,2}$ and $V_{3,3}$ to be ignored. Equation (7.22) can then be written as,

$$a'_V(z,\bar{z}) = -\int_{0}^{L} EA\big[z_{1,1}\bar{z}_{1,1}V_{1,1} + \big(z_{1,1}\bar{z}_{1,2} + z_{1,2}\bar{z}_{1,1}\big)V_{2,1} + \big(z_{1,1}\bar{z}_{1,3} + z_{1,3}\bar{z}_{1,1}\big)V_{3,1}\big]dx_1 \tag{7.38}$$

Since in a truss there are no distortions, the non-diagonal elements of the strain matrix are only related with rigid-body rotations and,

$$\begin{aligned} z_{2,1} &= -z_{1,2}\ and\ z_{3,1} = -z_{1,3} \\ \bar{z}_{2,1} &= -\bar{z}_{1,2}\ and\ \bar{z}_{3,1} = -\bar{z}_{1,3} \end{aligned} \tag{7.39}$$

Allowing rewriting the equation (7.38) in a way that is suitable for calculation, since the derivatives are now regarding only to the axis of the element,

$$a'_V(z,\bar{z}) = -\int_0^L EA\left[z_{1,1}\bar{z}_{1,1}V_{1,1} + \left(z_{1,1}\bar{z}_{2,1} + z_{2,1}\bar{z}_{1,1}\right)V_{2,1} + \left(z_{1,1}\bar{z}_{3,1} + z_{3,1}\bar{z}_{1,1}\right)V_{3,1}\right]dx_1 \tag{7.40}$$

7.3.5. Alpha Prime Calculation for the Beam2D Element

In the Beam2D element, besides the deformation due to axial loads, there is also deformation due to bending. The energy term of equation (7.9) due to bending is,

$$a(z,\bar{z}) \equiv \int_\Omega c(z,\bar{z})d\Omega = \int_0^L EIz_{3,11}\,dx_1 \tag{7.41}$$

Where E is the Young's modulus and I is the inertia of the cross-section regarding the X_2 axis. The elastic energy is expressed as function of the z_3 and $\bar{z}_3$ displacements, and of the components of the vectors $\nabla z^T V$ and $\nabla \bar{z}^T V$, that is, only $z_{3,1}V_1 + z_{3,2}V_2 + z_{3,3}V_3$ and $\bar{z}_{3,1}V_1 + \bar{z}_{3,2}V_2 + \bar{z}_{3,3}V_3$ are relevant, see figure 7.4. The four terms that make up equation (7.22) are,

$$\begin{aligned} c(\nabla z^T V,\bar{z}) = EI\big(&z_{3,111}V_1 + 2z_{3,11}V_{1,1} + z_{3,1}V_{1,11} + z_{3,211}V_2 + 2z_{3,21}V_{2,1} + z_{3,2}V_{2,11} \\ &+ z_{3,311}V_3 + 2z_{3,31}V_{3,1} + 2z_{3,3}V_{3,11}\big)\bar{z}_{3,11} \end{aligned} \tag{7.42}$$

$$\begin{aligned} c(z,\nabla \bar{z}^T V) = EI\big(&\bar{z}_{3,111}V_1 + 2\bar{z}_{3,11}V_{1,1} + \bar{z}_{3,1}V_{1,11} + \bar{z}_{3,211}V_2 + 2\bar{z}_{3,21}V_{2,1} + \bar{z}_{3,2}V_{2,11} \\ &+ \bar{z}_{3,311}V_3 + 2\bar{z}_{3,31}V_{3,1} + 2\bar{z}_{3,3}V_{3,11}\big)z_{3,11} \end{aligned} \tag{7.43}$$

$$\begin{aligned} \nabla c(z,\bar{z})^T V = EI\big[&\left(z_{3,111}\bar{z}_{3,11} + z_{3,11}\bar{z}_{3,111}\right)V_1 + \left(z_{3,112}\bar{z}_{3,11} + z_{3,11}\bar{z}_{3,112}\right)V_2 \\ &+ \left(z_{3,113}\bar{z}_{3,11} + z_{3,11}\bar{z}_{3,113}\right)V_3\big] \end{aligned} \tag{7.44}$$

$$c(z,\bar{z})\,divV = EI_2 z_{3,11}\bar{z}_{3,11}\left(V_{1,1} + V_{2,2} + V_{3,3}\right) \tag{7.45}$$

It is possible to assume [46],

$$\begin{gathered} z_{3,11i} = -z_{3,i11}, \\ z_{3,31} = z_{3,21} = \bar{z}_{3,31} = \bar{z}_{3,21} = 0 \\ V_{3,11} = V_{2,11} = 0 \end{gathered} \tag{7.46}$$

it is then possible to obtain,

$$\begin{aligned} c(\nabla z^T V,\bar{z}) + c(z,\nabla \bar{z}^T V) - \nabla c(z,\bar{z})^T V - c(z,\bar{z})\,divV \\ = EI\left[3z_{3,11}\bar{z}_{3,11}V_{1,1} + \left(z_{3,1}\bar{z}_{3,11} + z_{3,11}\bar{z}_{3,1}\right)V_{1,11}\right] \end{aligned} \tag{7.47}$$

The equation (7.22) can be then written, considering the effects of the axial and bending deformations, as,

$$a'_V(z,\bar{z}) = -\int_0^L EA\left[z_{1,1}\bar{z}_{1,1}V_{1,1} + \left(z_{1,1}\bar{z}_{2,1} + z_{2,1}\bar{z}_{1,1}\right)V_{2,1} + \left(z_{1,1}\bar{z}_{3,1} + z_{3,1}\bar{z}_{1,1}\right)V_{3,1}\right]dx_1$$
$$-\int_0^L EI\left[3z_{3,11}\bar{z}_{3,11}V_{1,1} + \left(z_{3,1}\bar{z}_{3,11} + z_{3,11}\bar{z}_{3,1}\right)V_{1,11}\right]dx_1 \quad (7.48)$$

7.3.5. Stress, Displacement and Volume Sensitivities

If a displacement in a point can be defined as,

$$\Psi = z_i(\hat{x}) = \int_\Omega \delta(x-\hat{x})\, z_i\, d\Omega \quad (7.49)$$

Where δ is the Dirac function, $\hat{x}$ is the position where the displacement z_i is measured. The first derivative of the expression (7.49) is,

$$\Psi' = \dot{z}_i(\hat{x}) = \int_\Omega \delta(x-\hat{x})\, \dot{z}_i\ d\Omega \quad (7.50)$$

the adjoint equation (7.27) becomes then,

$$a\left(\lambda,\bar{\lambda}\right) = \int_\Omega \delta(x-\hat{x})\, \bar{\lambda}_i\, d\Omega \quad for\ all\ \bar{\lambda} \in Z \quad (7.51)$$

The adjoint load from the right side of the equation (7.51) is a unitary load applied on the $\hat{x}$ point in the z_i direction. Once the adjoint solution λ is found, the sensitivities can be calculated through the equation (7.32), which for the displacement performance can be rewritten as,

$$\Psi' = l'_v(\lambda) - a'_v(z,\lambda) \quad (7.52)$$

In its turn, a stress can be defined into a point as,

$$\Psi = \int_\Omega g\left(\sigma(z)\right) m_p\, d\Omega \quad (7.53)$$

where g is the function of the components of the stress matrix σ and m_p is a function which presents a null value for all the points of the domain, except for the subdomain Ω^i where is intended to evaluate the stress, whose integral is $1/\Omega^i$. To calculate the average stress in Ω^i, equation (7.53) becomes,

$$\Psi = \int_{\Omega^i} g\left(\sigma(z)\right) m_p\, d\Omega \Big/ \int_{\Omega^i} d\Omega \quad (7.54)$$

the first derivative of the expression (7.53) is,

$$\Psi' = \int_{\Omega} \left(g' + \nabla g^T V + g(divV)\right) m_p \, d\Omega$$
$$- \int_{\Omega} g \, m_p \, d\Omega \cdot \int_{\Omega} (divV) \, m_p \, d\Omega \tag{7.55}$$

To calculate the average stress in the domain Ω^i, the first derivative of the expression (7.54) is,

$$\Psi' = \frac{\left[\int_{\Omega^i}\left(g' + \nabla g^T V + g(divV)\right) d\Omega \cdot \int_{\Omega^i} d\Omega - \int_{\Omega^i} g \, d\Omega \cdot \int_{\Omega^i} divV \, d\Omega\right]}{\left[\int_{\Omega^i} d\Omega\right]^2} \tag{7.56}$$

The adjoint equation (7.27) for the stress performance becomes,

$$a(\lambda, \bar{\lambda}) = \int_{\Omega^i} g_{\sigma_{jk}} \sigma_{jk}'(\bar{\lambda}) \, d\Omega \quad for\ all\ \bar{\lambda} \in Z \tag{7.57}$$

For the stress performance, the adjoint load in the right side of the equation (7.57) represents the set of loads necessary to apply to get a unit state of stress in the element. The sensitivities can be calculated through equation (7.32), which for the stress performance can be rewritten as,

$$\Psi' = \int_{\Omega} \sum_{j,k=1}^{3} g_{\sigma_{jk}}(z) \left[\sigma_{jk}(\dot{z}) - \sigma_{jk}(\nabla z^T V)\right] m_p \, d\Omega$$
$$+ \int_{\Omega} \sum_{m=1}^{3} \left[\sum_{j,k=1}^{3} g_{\sigma_{jk}}(z) \, \sigma_{jk,m}(z) V_m\right] m_p d\Omega + \int_{\Omega} g \, (divV) \, m_p d\Omega$$
$$- \int_{\Omega} g \, m_p d\Omega \cdot \int_{\Omega} g \, (divV) \, m_p d\Omega \tag{7.58}$$

considering,

$$\sigma_{jk}(\nabla z^T V) = \sum_{m,n=1}^{3} C_{jkmn}\left(\nabla z_{m,n}^T V + \nabla z_m^T V_{,n}\right) \tag{7.59}$$

and,

$$\sum_{m=1}^{3} \sigma_{jk,m}(z) V_m = \sum_{m,n=1}^{3} C_{jkmn}\left(\nabla z_{m,n}^T V\right) \tag{7.60}$$

the equation (7.58) becomes,

$$\Psi' = \int_{\Omega} \sum_{j,k=1}^{3} g_{\sigma_{jk}}(z) \, \sigma_{jk}(\dot{z}) \, m_p \, d\Omega - \int_{\Omega} \sum_{j,k=1}^{3} \left[g_{\sigma_{jk}}(z) \sum_{m,n=1}^{3} C_{jkmn}\left(\nabla z_m^T V_{,n}\right)\right] m_p d\Omega$$
$$+ \int_{\Omega} g \, (divV) \, m_p d\Omega - \int_{\Omega} g \, m_p d\Omega \cdot \int_{\Omega} g \, (divV) \, m_p d\Omega \tag{7.61}$$

Finally using the adjoint method,

$$\Psi' = l'_v(\lambda) - a'_v(z,\lambda) - \int_\Omega \sum_{j,k=1}^{3} \left[g_{\sigma_{jk}}(z) \sum_{m,n=1}^{3} C_{jkmn}\left(\nabla z_m^T V_{,n}\right) \right] m_p d\Omega \qquad (7.62)$$

The last performance, the volume performance, can be defined as,

$$\Psi = \int_\Omega d\Omega \qquad (7.63)$$

and its first derivative is,

$$\Psi' = \int_\Omega (divV)\, d\Omega \qquad (7.64)$$

The sensitivity calculation can be decomposed into 4 stages, as it is shown in the figure 7.5. The solution of the original problem enables to know the displacement field of the original structure. The solution of the adjoint problem allows, for each performance, to obtain the adjoint load vector in equation 7.27. The velocity field translates the relationship between the change of the variable which controls the shape and the corresponding change of position by all the points of the domain. Finally the sensitivities can be calculated by numerical integration of the expression 7.32.

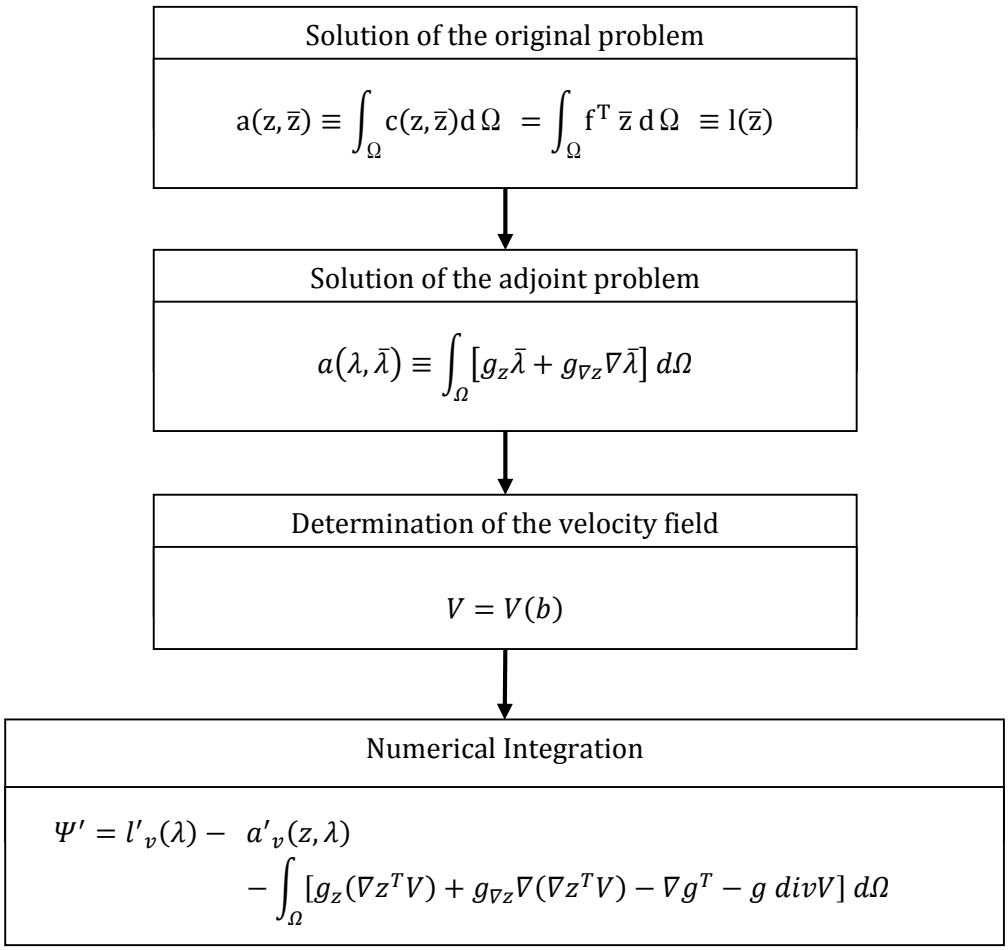

Figure 7.5- Stages in shape design sensitivitiy calculation

Chapter 8

8. EFFECT

8.1. Overview

From the start, it was decided that EFFECT wasn't just going to be a regular finite element program, solely capable of solving the typical linear and nonlinear static and dynamic problems; it would also be able to perform sensitivity analysis. Knowing that the implementation of a full set of finite elements and analysis types would be excessively time consuming, the emphasis was mainly put on the design of a framework to be easily extended later on, through the addition of new elements and capabilities, than in the construction of a traditional finite element program. This framework presents several innovative features that include the interfaces required to allow EFFECT to be integrated into a system capable of performing structural optimization and the possibility of computing precise values for the derivatives of specific finite elements quantities, such as nodal displacements and element stresses with respect to design variables, with the continuum method of design sensitivity analysis. Also, it was decided that EFFECT's input file format should be designed in order to be compatible with the one used by ANSYS, allowing the content of any file read by EFFECT to also be read by ANSYS. The greatest benefit of this kind of compatibility is the possibility of solving the same problem with both finite element programs using the same input file, reducing the effort involved in the preparation of these files. It also has the advantage of allowing the use of CAD systems and other FE programs that are able to write data in ANSYS format.

The developed framework includes 6 different structural finite element types (Truss2D, Truss3D, Beam2D, Beam3D, Node3Shell, and Node4Shell) and the possibility to solve static linear and nonlinear analysis and modal analysis. But the 6 element types only are available to perform static linear analysis, as nonlinear and modal analysis can only be performed on models using Beam2D elements. Also, the derivatives obtained with the continuum method of design sensitivity analysis can only be computed with Truss2D and Beam2D finite elements.

This chapter is intended to describe the proposed framework and should be helpful to anyone interested in understanding how EFFECT works. First, EFFECT architecture is shown, then EFFECT's execution is studied and finally a detailed analysis of its running behavior, explaining the interaction of the classes and objects is presented.

8.2. EFFECT's Architecture

The first step in the development of any software should be the design of its architecture and this is especially true when object oriented (OO) approaches are applied. These methodologies have specific features, such as classes, objects, inheritance, virtual methods and data abstraction that favor high-level conceptual design of software. A well thought out architecture, from the beginning, allows the smooth evolution of the program minimizing future code inconsistencies. It was decided that EFFECT's classes should be divided according to their purpose on the global scheme, the result was a structure with 8 classes that are derived from the basic *EFFECT* class: (1) *Sensitivity*; (2) *InOutputManager*; (3) *GaussPoint*; (4)

SolutionManager; (5) *Model;* (6) *NumericalMethod;* (7) *AlgebraicClasses;* (8) *StructuralComponents*. The only classes that are not inherited from these 8 are the ones used in the interface, and since they were only designed to improve the interaction with the program and do not participate in the main computation algorithms, these interface classes will not be described here.

Although different options to design EFFECT architecture could have been taken, it was decided to follow some basic rules, such as defining the node, element, material as basic objects as recommended in the work of Forde *et al.* [31], and the separation of the FEM Objects from the algebraic features as found in the work of Zimmermann *et al.* [32]. The code in general was organized in a highly intuitive fashion, allowing a quick understanding of the basics of EFFECT, just by studying its architecture diagram shown in the figure 8.1.

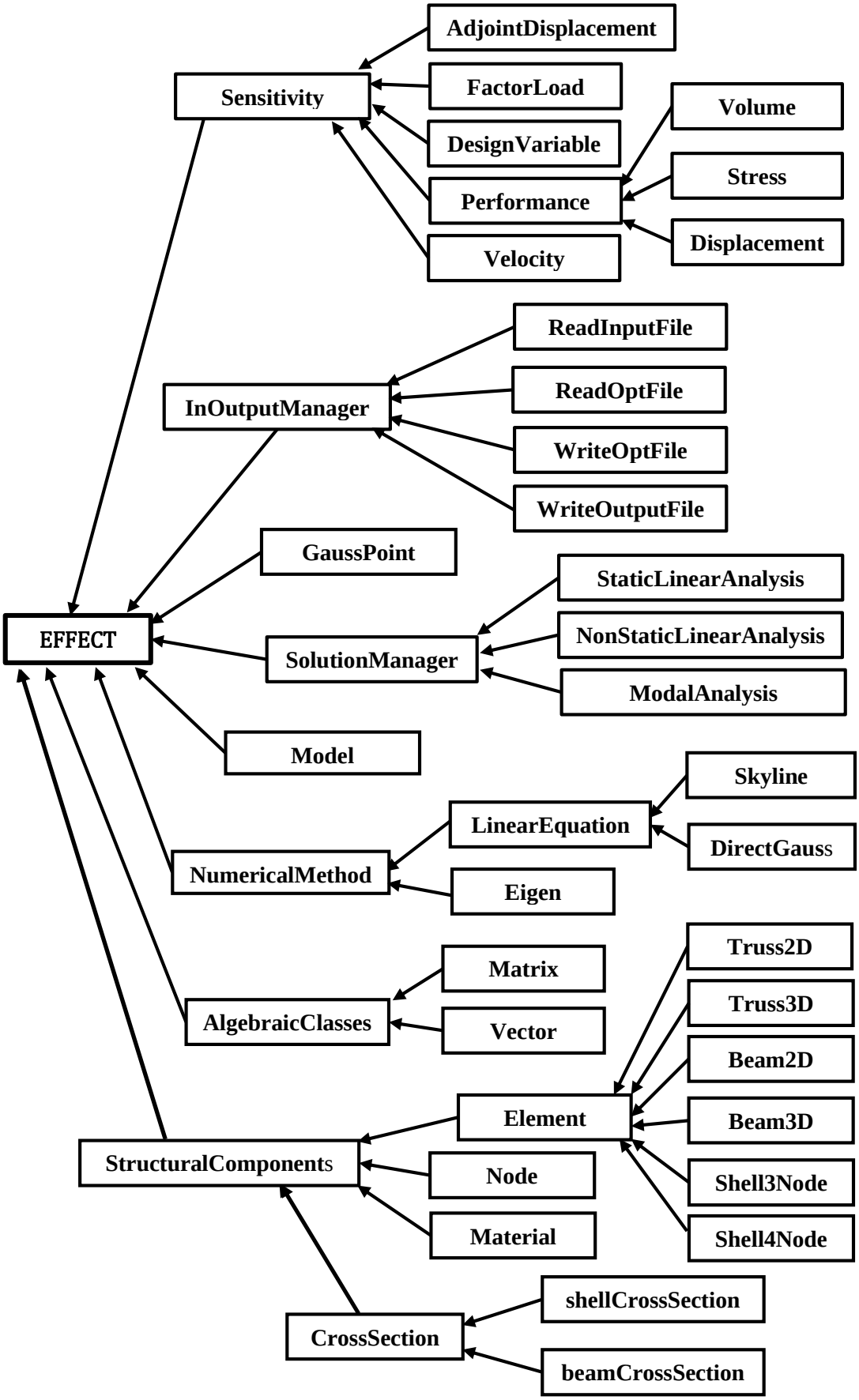

Figure 8.1 - EFFECT's architecture.

8.3. EFFECT's Control Systems

The interaction of the user with EFFECT was one of the first concerns taken in account, the intention was to design the software to be as user friendly as possible, but also practical for all types of usage. To fulfill those objectives 2 different type of interfaces were created: (1) A command console interface, managed by the *EFFECTConsole* class; (2) a simplified graphical interface, managed by the class *EFFECTInterface* class.

Both the console and the graphical interfaces are pretty self-explanatory, they were designed to be as simple as possible, making them very easy to use and allowing any user to quickly begin to interact with EFFECT.

Figure 8.2 show the main menu of the EFFECT console interface, running in a command window of the Windows operating system, with all the available command options:

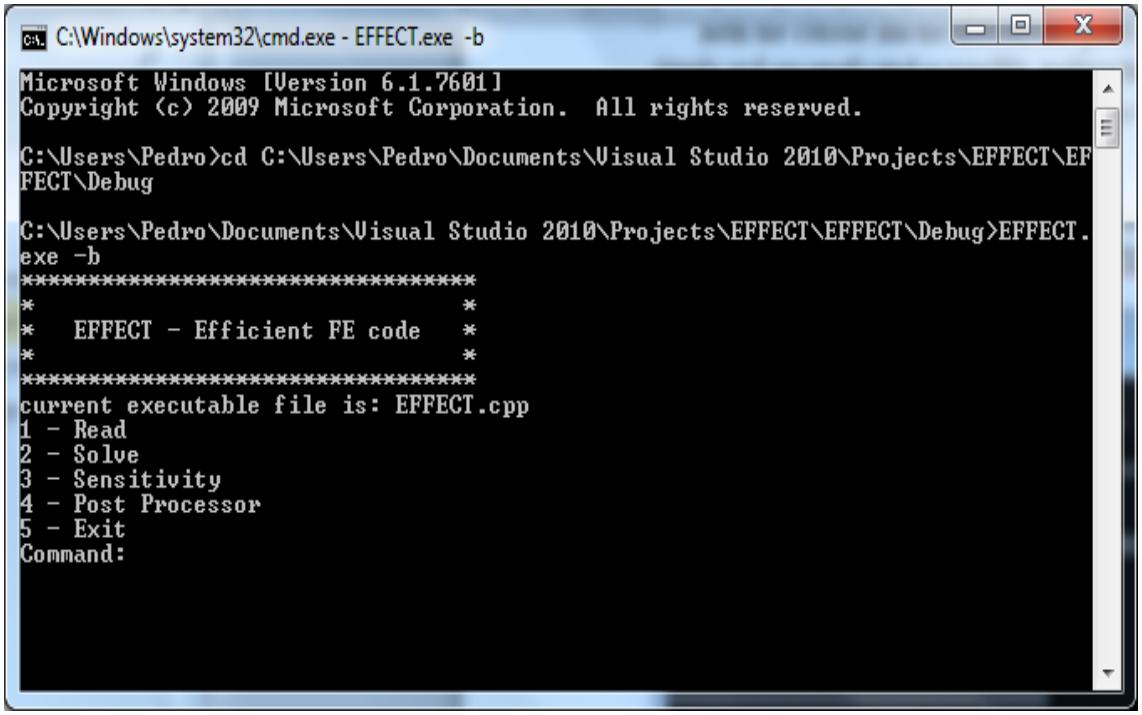

Figure 8.2 - EFFECT console interface running on the command window.

The menu options shown in figure 8.2 perform the following tasks:

(1) Read – initiates the analysis of a new structural problem. The *EFFECTConsole* class creates new instances of the *Model* and the *ReadInputFile* classes. The *ReadInputFile* class reads the input data file and stores the information in the *Model* class.

(2) Solve – initiates the procedure to solve the problem stored in the *Model* class. The *EFFECTConsole* class calls out the *Model* class, to decide which one of the derived classes from *SolutionManager* base class will handle the request to find a solution, based on the information stored in *Model*.

(3) Sensitivity – initiates the design sensitivity analysis. The *EFFECTConsole* class calls out the *Model* class, which together with the derived classes of the *Sensitivity* class

executes the task of calculating the derivatives of the required performances with respect to the assigned design variables.

(4) Post Processor – The *EFFECTConsole* class calls out the *WriteOutputFile* class to write the results of the structural analysis stored in the *Model* class.

(5) Exit – Closes EFFECT program and exits.

Figure 8.3 shows EFFECT graphical interface managed by *EFFECTInterface*. The options available in the interface behave exactly as the ones described for the console.

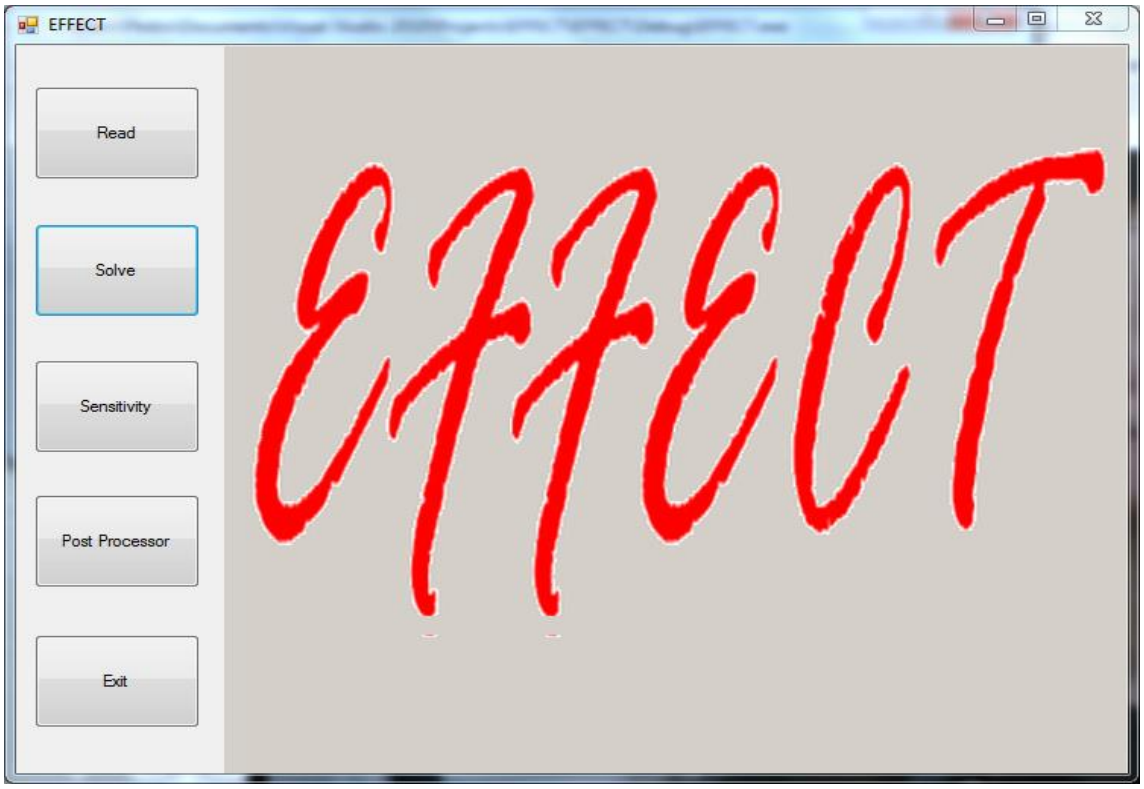

Figure 8.3 - EFFECT graphical interface.

The specific interface activated depends entirely in the method used to execute EFFECT. If the executable file name *effect.exe* is passed to the operating system, the graphical user interface is initiated by default. To execute the console interface, flag *–b* should be appended to EFFECT file name, as described in the user's manual.

The two interfaces presented are in fact redundant, because it is always possible to read the input data file, execute the analysis and write the results file without displaying any interface. Class *ReadInputFile* is prepared to run the analysis solely with the information received through the input file. For this purpose, all the analysis options and parameters must be included in this file, i.e., the input data file must contain all the information regarding the structure and all the data needed to control the execution process. All the commands available to write this input file are described in great detail on the EFFECT's manual.

8.4. EFFECT's Execution Behavior

After defining the class hierarchy and specifying the interfaces, the next challenge was to define the EFFECT's execution behavior. That encompasses the definition of the procedures or methods that allow the classes to interact with each other and the workflow for the various analysis types. It was also necessary to decide about EFFECT's capabilities and limitations, since the time schedule that was available to completely develop the software was rather short, it was important to carefully assess priorities, for instance the sensitivities analysis was only implemented on 2 elements: Truss2D, Beam2D elements.

Taking in consideration EFFECT's capabilities, it was chosen to divide EFFECT's execution behavior into three different modes: **(1)** Structural analysis Mode; **(2)** Sensitivity analysis mode; **(3)** Optimization mode. A careful analysis of these three modes will be done here.

The best method to illustrate a process is undoubtedly through diagrams, so in order to distinguish different objects in EFFECT, the following convention was chosen:

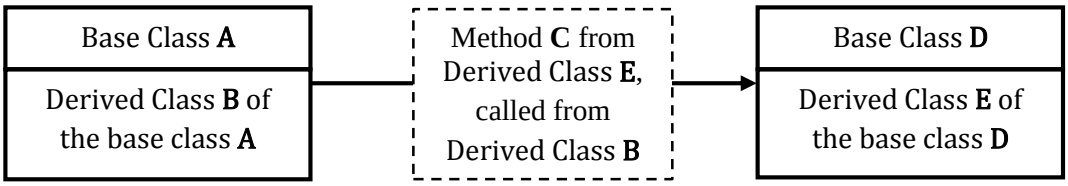

Figure 8.4 – Adopted convention to illustrate classes and methods interaction.

An object-oriented programming language, such as C++, has the possibility of using virtual methods. So in the case of calling a method, whose behavior is overridden within the derived classes of the base class in which it is declared, the convention is as shown in figure 8.5.

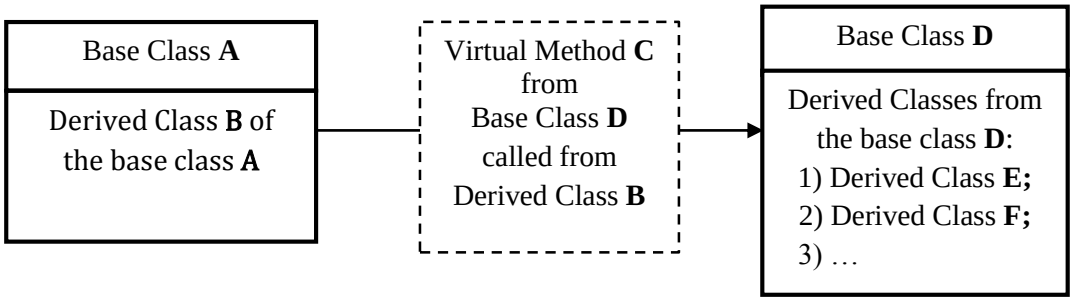

Figure 8.5 - Adopted convention in case of virtual methods.

In other words, whenever the virtual method "**C**" is called, what is actually being called is a method with the same signature (name), from one of the derived classes of the base class "**D**". In the figure 8.6 is shown the convention used to illustrate when a method calls another method in the same class.

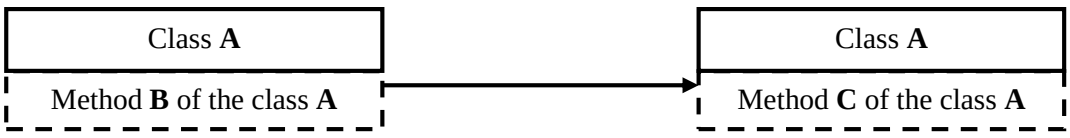

Figure 8.6 - Adopted convention for calling a method in the same class.

8.4.1. Structural Analysis Mode

The structural analysis process begins with the user command for reading a new input file. This command not only creates an instance of the *ReadInputFile* class, derived from *InOutputManager* class, but also a new instance of the *Model* class, which is passed as argument to the *ReadInputFile* class.

The purpose of the *ReadInputFile* class is to handle the input data file. The *ReadInputFile* class reads and stores all relevant information regarding the structure in the *Model* class and executes any commands that may be stated in this file. After the file has been properly read there are two possibilities, depending on the instructions it contains. If no explicit command is given about how EFFECT should proceed, then the control system (*EFFECTConsole* or *EFFECTInterface* class) initially used takes back the control of the execution process and waits for the user input. However, if an analysis command is present in the file, then the *ReadInputFile* class automatically calls out the *executeAnalysis* method of the *Model* Class, through the pointer which was passed as argument, and execution process proceeds in the *Model* class. Regardless of where the analysis command came from, the *executeAnalysis* method creates an instance of one of the following classes derived from the *SolutionManager* class:

- *StaticLinearAnalysis* class;
- *NonLinearStaticAnalysis* class;
- *ModalAnalysis* class.

Which of these classes is actually created, obviously depends on the type of analysis the user wishes to perform, and such information is stored in the *analysisType* member. This member is set to 0 by default, which is equivalent to a static linear analysis and therefore, if no analysis type is specified, EFFECT automatically creates an instance of the *StaticLinearAnalysis* class, replicating ANSYS behavior. The value of *analysisType* can be altered through the input file, the console or the graphical interfaces. This issue is fully covered in EFFECT's user manual.

To actually solve the problem, *executeAnalysis* calls the *computeAnalisys* virtual method that exists in the instance of the derived class of *SolutionManager*. Since *computeAnalisys* is a virtual method all three derived classes use the same signature, but the code of the method can be overridden in each class, in other words, all three classes have a method called "*computeAnalisys",* but it performs different tasks in each of them. The execution is now passed to one of the derived classes of the *SolutionManager* class, which at this point calls the methods of the *NumericalMethod* class to assist solving the equations required. There are two classes directly derived from the *NumericalMethod* class: (1) *LinearEquations* class; (2) *Eigen* class. The *Eigen* class is called directly from the *ModalAnalysis* class, but the *LinearEquations* class has also two derived classes, *Skyline* and *DirectGauss*, which are called from *StaticLinearAnalysis* or *NonLinearStaticAnalysis* classes.

After finding the solution it only remains to write the results. Again there are two possibilities, if the analysis command was present in the data file, EFFECT will automatically create an instance of the class *WriteOutputFile* derived from the *InOutputFile* class to save the results in a file. If this is not the case, the user is prompted with the menu and should give the respective command. The workflow of the structural analysis mode is illustrated in the figure 8.7.

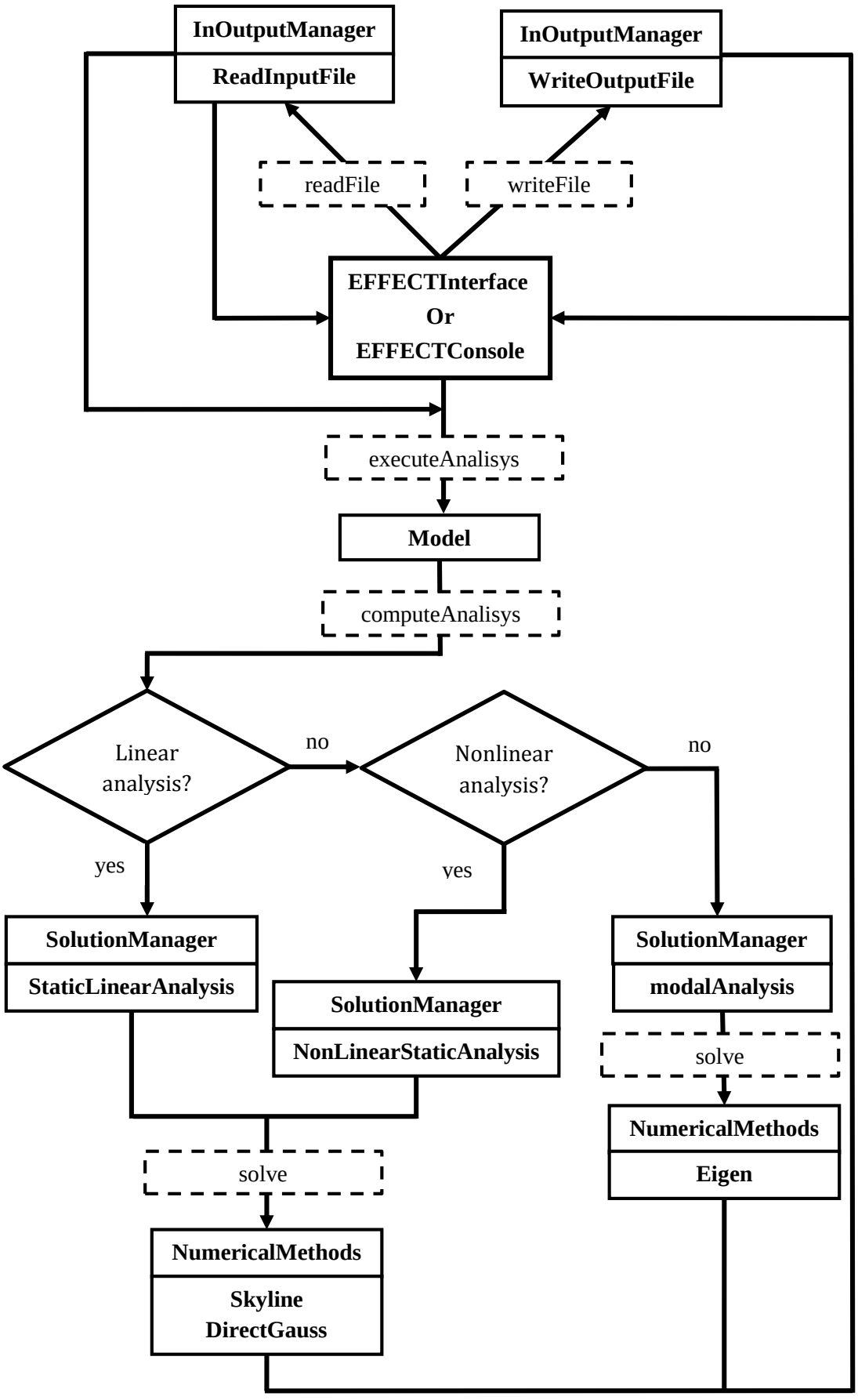

Figure 8.7 – EFFECT's execution process in the structural analysis mode.

8.4.2. Sensitivity Analysis Mode

When EFFECT is executed in this mode it will perform a sensitivity analysis and compute the performance gradients with respect to the design variables, for all performances and variables defined in *Model*.

As in the first mode, when the *ReadInput* class finishes reading the data file, there are also two possibilities; the command for the calculation of sensitivities is contained in the input file or it is specified through the user interface. Regardless of how the command is stated, the *Model* class *executeSensitivityAnalysis* method is immediately called. This method then calls the three methods necessary to compute sensitivities:

1. *executeAnalysis* – Does a normal linear static analysis;
2. *executeStructAdjointReanalysis* – calculates the adjoint displacements;
3. *executeStructAdjointSensitivity* – evaluates the sensitivities expressions.

After *executeAnalysis* and *executeStructAdjointSensitivity* methods have done their tasks and both the original and the adjoint displacements have been calculated and stored, the *executeStructAdjointSensitivity,* is called to initiate the sensitivities calculations. Of course those calculations are different depending on the specific performance, design variable and finite element used. Since it is possible to calculate sensitivities regarding three types of performances, there are three different classes derived from the *Performance* class to represent each one of them: (1) *Stress* class; (2) *Displacement* class; (3) *Volume* class.

As an example, if two performances are declared in the input file, one of the stress type and the other of the displacement type, then one instance of the *Stress* Class and the other of the *Displacement* Class are created and stored in the *Model* class, during the file reading process.

The *executeStructAdjointSensitivity* method will be studied more carefully later on, but its overall purpose is to send every single instance stored of the *Element* class to the *doAdjointSensitivity* virtual method of the *Performance* class. Once again, because the *doAdjointSensitivity* is a virtual method every derived class can call it, even though it performs different tasks for each of them. The purpose of the *doAdjointSensitivity* is to identify the type of performance in question and then call the correct methods from the *Element* class to proceed to the sensitivity calculation. The methods that *doAdjointSensitivity* calls for the different performances are:

- *Stress* class – *doAprime* virtual method of the *Element* class;
- *Displacement* class – *doAprime* and *doVonMisesPrime* virtual methods of the *Element* class;
- *Volume* class – *doMaterialVolumePrime* virtual method of the *Element* class.

As in the structural analysis mode, after the sensitivity analysis is completed and if the analysis command was present in the data file, an instance of the *WriteOutputFil*e class, derived from *InOutputManager* class, is automatically created to save the results in a file. The execution process of the sensitivity analysis mode is illustrated in the figure 8.8 and 8.9.

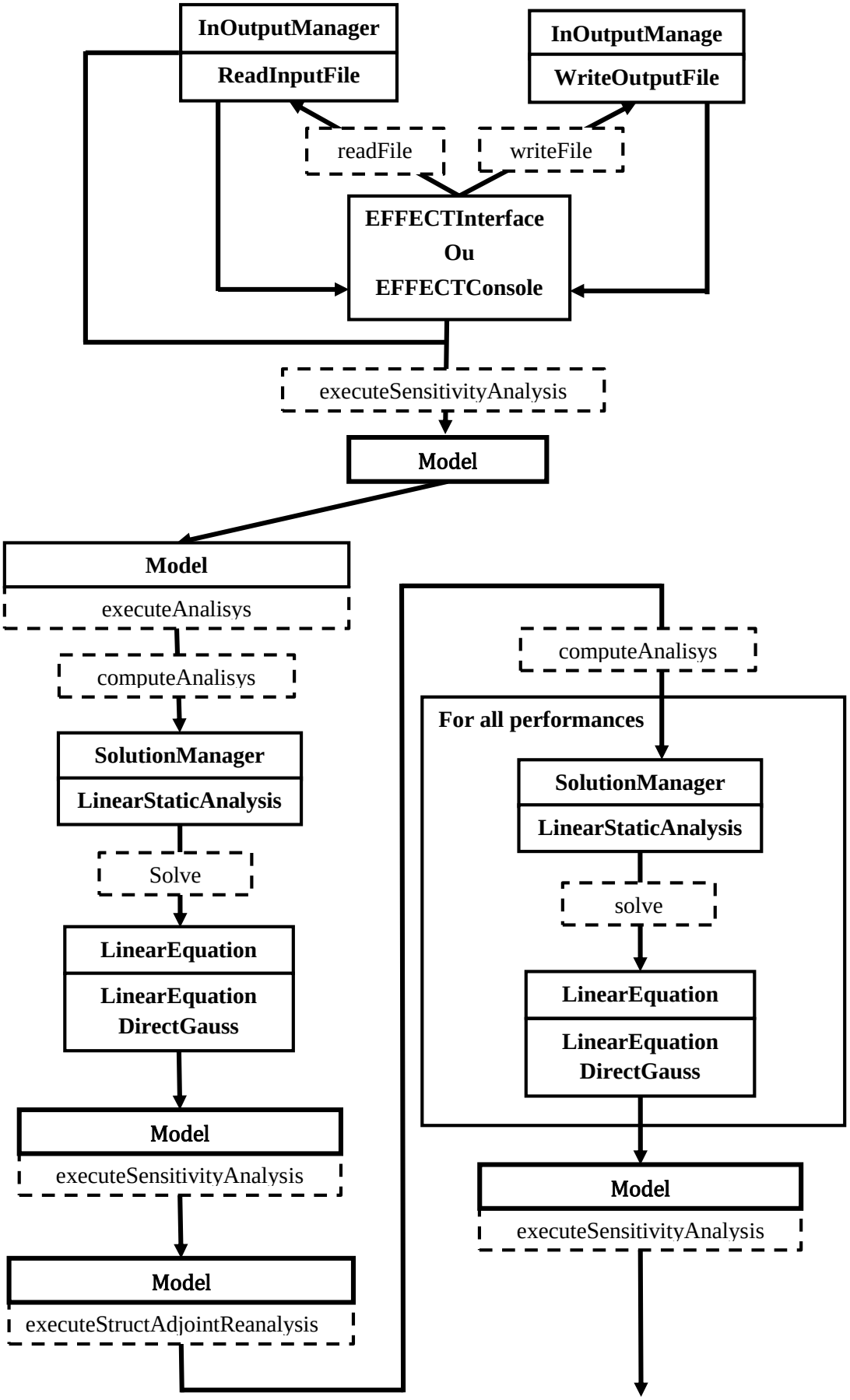

Figure 8.8 – EFFECT's execution process in the sensitivity analysis mode.

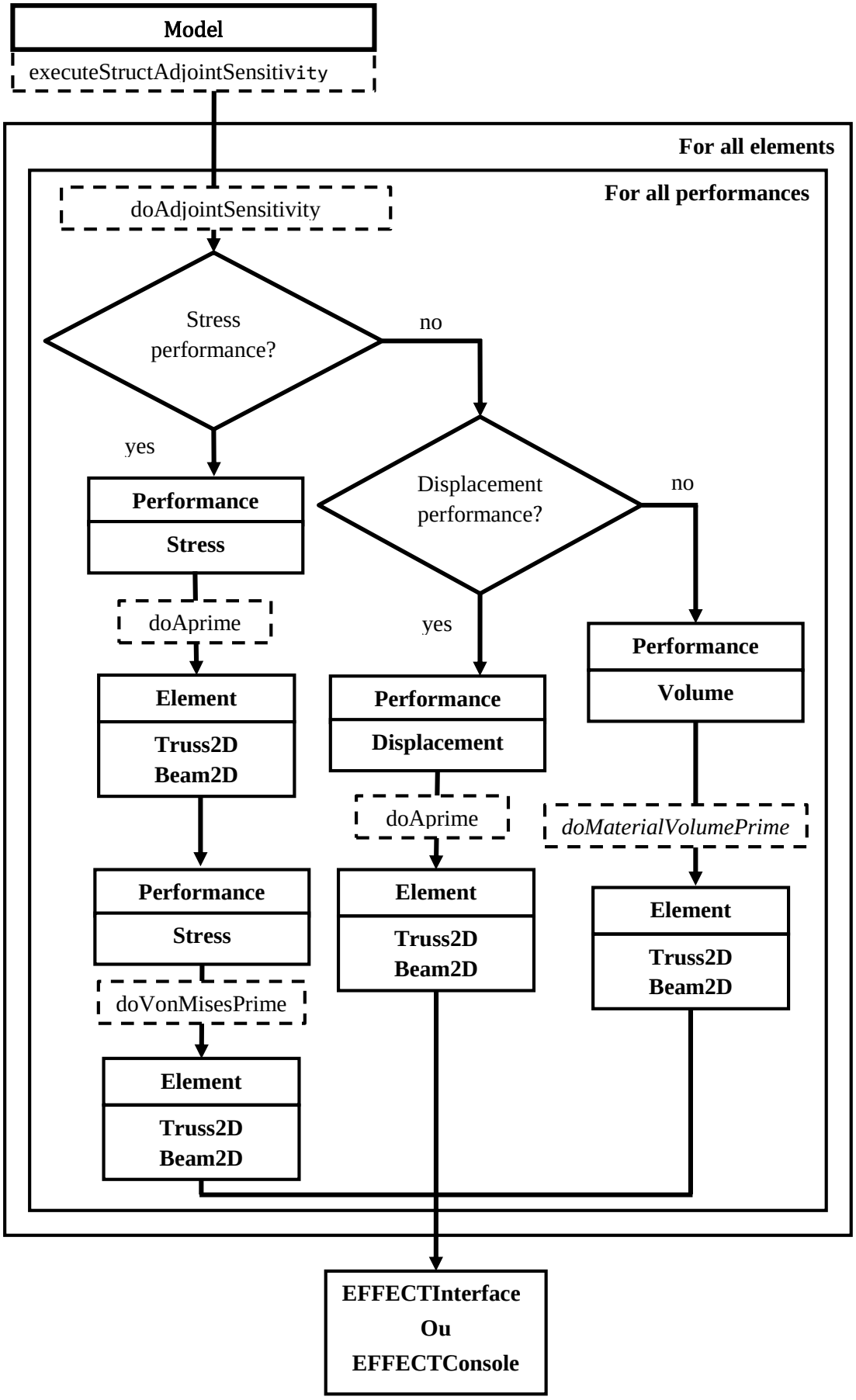

Figure 8.9 - Continuation of the execution process diagram of the sensitivity analysis.

8.4.3. Optimization With MATLAB

EFFECT is prepared to be included into a software system to obtain the solution of structural optimization problems. For that purpose, interfaces allowing information related to design variables and objective or constraint functions have been created. The program also computes precise values of gradients of these functions, which can be usefully used by some optimization algorithms. EFFECT does not include these algorithms, and relies on external tools, such as the optimization toolbox of the commercial software MATLAB, *optimtool*. This tool requires the definition of two functions, one to set the objective function, **myfunc.m**, and the second to define the constraints, **mycon.m**.

The interaction between EFFECT and MATLAB is based on a simple mechanism of file exchange, the *opt2effect* and *effect2opt* files. The *opt2effect*, is responsible for transmitting the values of the design variables to EFFECT, for each iteration. This file is read by the *ReadOptFile* class derived from *InOutputManager*. This class receives the instance of the

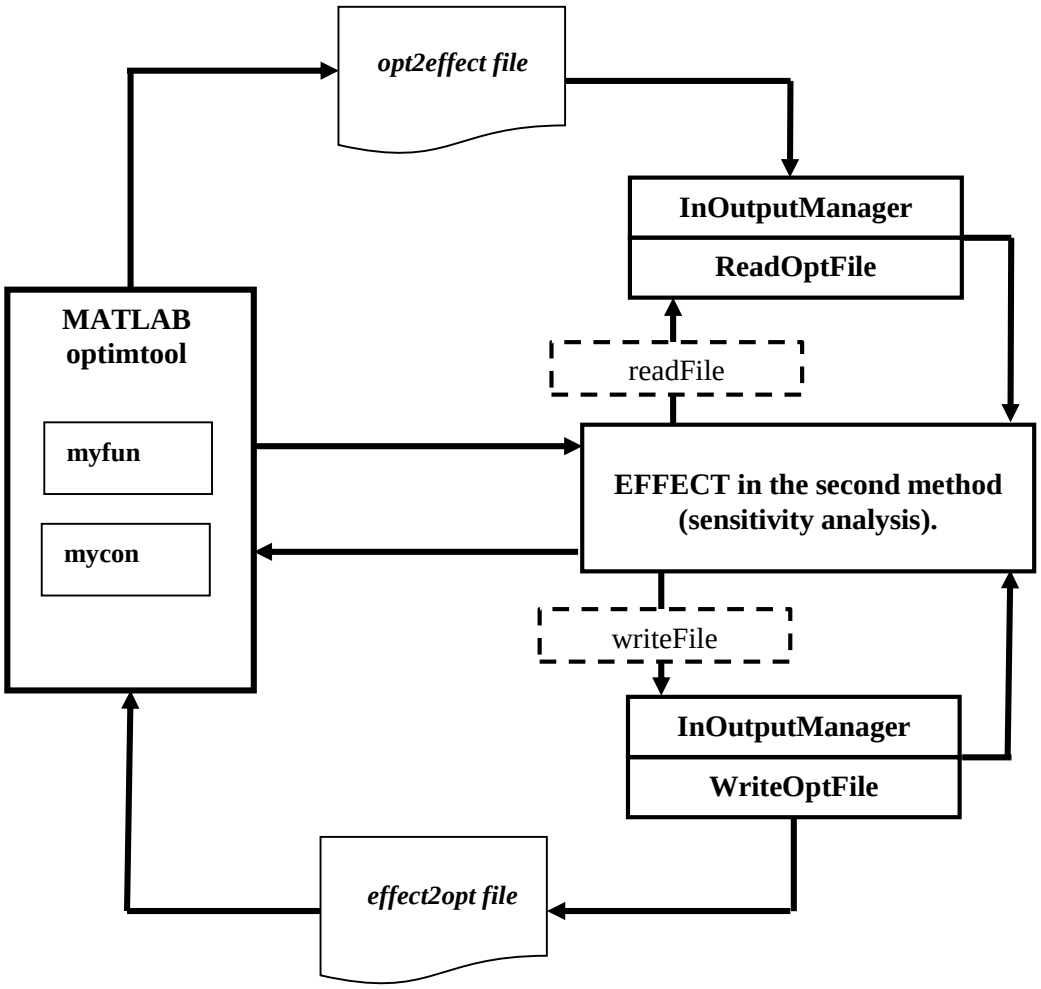

Figure 8.10 – EFFECT's execution process in the optimization mode.

Model class, so during the reading process the values of the variables stored in *Model* can be updated immediately. The *effect2opt* file is written by the *WriteOptFile* class, also derived from *InOutputManager* and its purpose is to deliver the values of the functions that describe

structural behavior (displacements at nodes, stresses at elements, volume) and that are used in the definition of the objective and constraint functions. These functions are here referred as performances. The *effect2opt* file can also transmit the performance gradients regarding the defined design variables to the *optimtool* of MATLAB.

When solving an optimization problem, EFFECT is executed from inside **myfunc.m**, or **mycon.m** MATLAB functions in optimization mode. The optimization process begins in the *optimtool*, that typically requests EFFECT to compute functions and eventually also gradients, for each iteration of the optimization algorithm. When EFFECT is initiated in the optimization mode it becomes fully automatic, it first searches for the *opt2effect* file as shown in the figure 8.10 and then computes performances and sensitivities without any user intervention. After the structural and sensitivity analysis has been completed EFFECT writes the results in the *effect2opt* file for the optimization algorithm and terminates execution. No console or graphical interfaces are used in this mode.

8.5. EFFECT's Classes

Once the EFFECT global structure, the interfaces and the behavior for the three different execution modes are described, it is important to study more carefully the building blocks of the program, EFFECT's classes. As illustrated in figure 8.1, EFFECT's architecture was divided into eight main groups of classes: (1) *Sensitivity*; (2) *InOutputManager*; (3) *GaussPoint*; (4) *SolutionManager;* (5) *Model;* (6) *NumericalMethod;* (7) *AlgebraicClasses;* (8) *StructuralComponents* class.

These classes are now going to be described with different levels of attention according to their importance, complexity or function. The *GaussPoint* and *AlgebraicClasses* classes won't be discussed because their functions are simple and clear. The *GaussPoint* stores the information required for the evaluation of integral expressions. The *AlgebraicClasses* holds all the algebraic operations with matrices and vectors.

8.5.1. Model Class

The *Model* class has a central role in EFFECT as it already may have been perceived in the execution diagrams. The importance of the *Model* class lies in two different aspects, on the one hand it is the storage place of the data regarding any problem to be solved, and on the other hand it represents the domain of the problem, which makes that every operation that EFFECT executes regarding any analysis has to go through *Model*.

The data storage takes place during the reading process of the input file, in the *ReadInputFile* class. Whenever a specific type of data (e.g. a node, an element, etc.) is defined in this file, a new instance of a class, that represents that type of information, is created. For example, if in the input file appears a command instructing to create a node then a new instance of the class *Node* is created. However, in any given problem, thousands of nodes, elements, cross sections and materials may be defined, and luckily the C++ language provides a library of classes that can dynamically allocate space for each object to be stored. So when an instance or object of a class is created it is then immediately saved in the respective list for the objects of its type. *Model* holds six different lists, which means it is prepared to store the objects of six different classes; those classes are represented in figure 8.11 and shown in table 8.1.

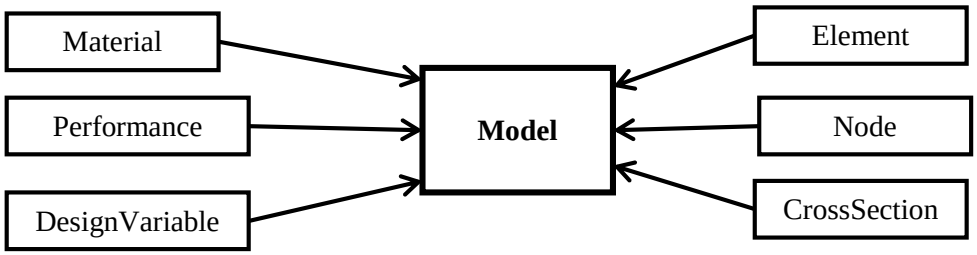

Figure 8.11 - Classes which are contained in *Model* lists.

Table 8.1 - Lists of the *Model* class.

***Model*'s Lists**	**Description**
std::list <Node *> *nodeList*	Stores all the objects of the *Node* class type.
std::list <Material *> *materialList*	Stores all the objects of the *Material* class type.
std::list <Element *> *elementList*	Stores all the objects of the *Element* class type.
std::list <CrossSection *> *crossSectionList*	Stores all the objects of the *cross Section* class type.
std::list < DesignVariable *> *designVariableList*	Stores all the objects of the *DesignVariable* class type.
std::list < Performance *> *performanceList*	Stores all the objects of the *Performance* class type.

While the *CrossSection*, *Material*, *DesignVariable* and *Node* classes only exist to store information, the *Element* and *Performance* classes also perform different tasks, as it was seen during the explanation of EFFECT's execution process. In table 8.2 the remainder of the more relevant members of the *Model* class are shown.

Table 8.2 - Members of the *Model* class

***Model'*s Members**	**Description**
int *numberOfEquations*	Stores the number of active degrees of freedom on the structure.
int *analysisType*	Identifies the type of analysis intended to perform
int *nonLinearAnalysis*	Identifies if the type of analysis intended to perform is nonlinear or not
bool *skylineStatus*	Identifies if the skyline method to solve the equations is on or off.
int *numberOfDesignVariable*	Stores the total number of Design variables.
int *numberOfShapeVariable*	Stores the number of shape variables.
int *numberOfPerformance*	Stores the number of Performances.

To finish with the *Model* class it is important to describe its methods, which are divided in 2 different tables, table 8.3 for general methods and table 8.4 for the sensitivity analysis related methods.

Table 8.3 - General methods of the *Model* class.

***Model's* Methods**	**Description**
Model(void)	Constructor.
bool *getSkylineStatus* (void)	Returns the *skylineStatus* member.
void *changeNonLinearAnalysis*(int **option**)	Changes the value of *nonLinearAnalysis* member to the value of the option variable, received as parameter.
void *computeTotalNumberEquations*(void)	Computes the total number of active degree of freedoms for the entire structure.
void *setSkylineOFF* (void)	Changes the value of *skylineStatus* member to **false (**by default it is set to **true).**

Table 8.4 - Sensitivity related methods of the *Model* class.

Model's **Methods**	**Description**
bool *executeSensitivityAnalysis*(void)	Calls out the *Model*'s methods which take part on the sensitivity analysis. Returns **true,** if the analysis was successful.
void *executeStructAdjointReanalysis* (void)	Initiates a new static linear analysis for the adjoint displacements calculations.
void *executeStructAdjointSensitivity* (void)	Sends the instances of the *Element* class to the instances of the *Performance* class, to initiate the sensitivity calculations.

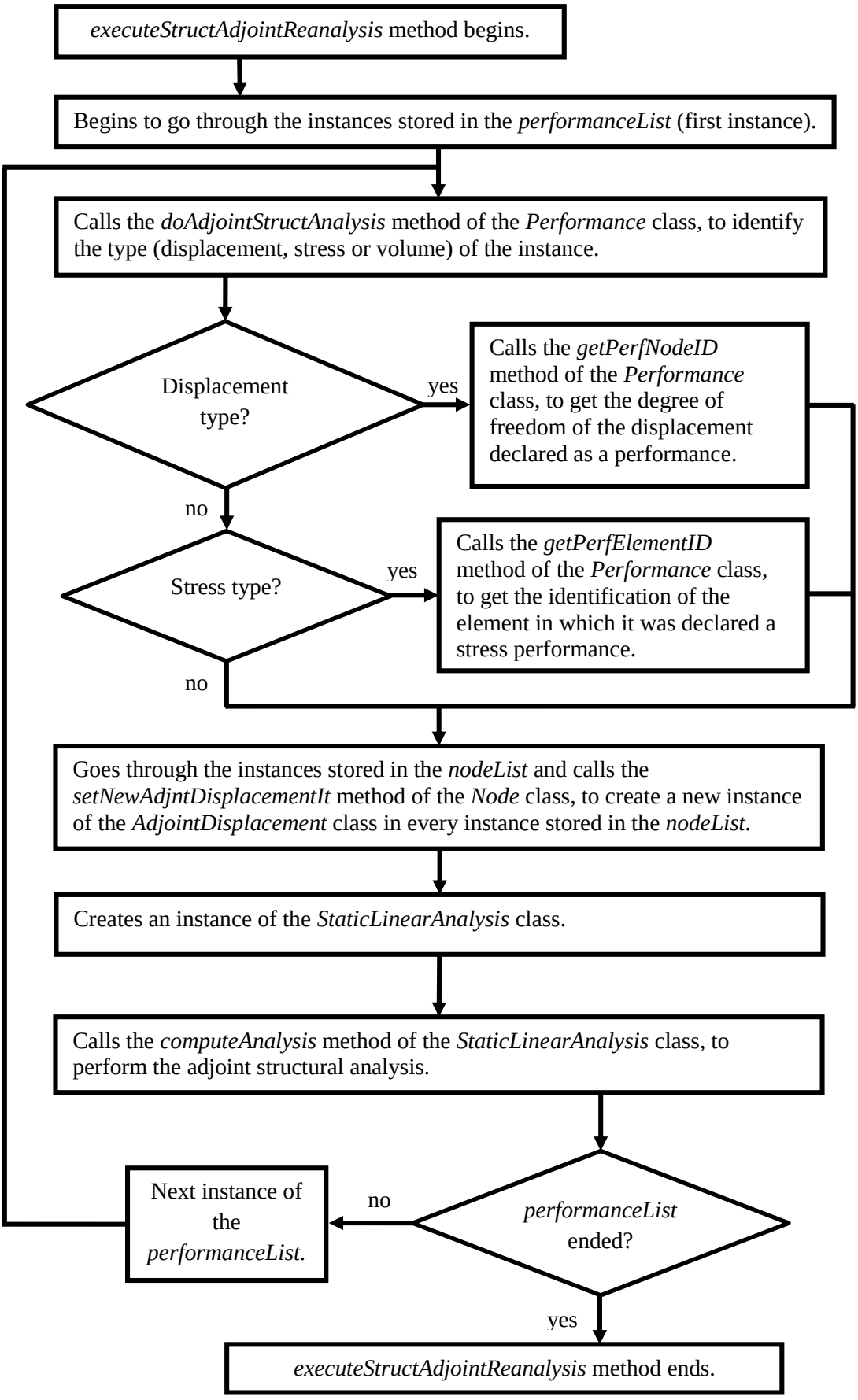

Figure 8.12 – Execution process diagram of the *executeStructAdjointReanalysis* method.

The *executeStructAdjointReanalysis* and the *executeStructAdjointSensitivity* methods were already shown in the diagram of figure 8.9 describing the overall process of executing a sensitivity analysis. However, due to their importance and complexity it is worth to make a more careful analysis of both methods. The *executeStructAdjointReanalysis* execution process can be seen in the figure 8.12 and the one of *executeStructAdjointSensitivityin* in the figure 8.13.

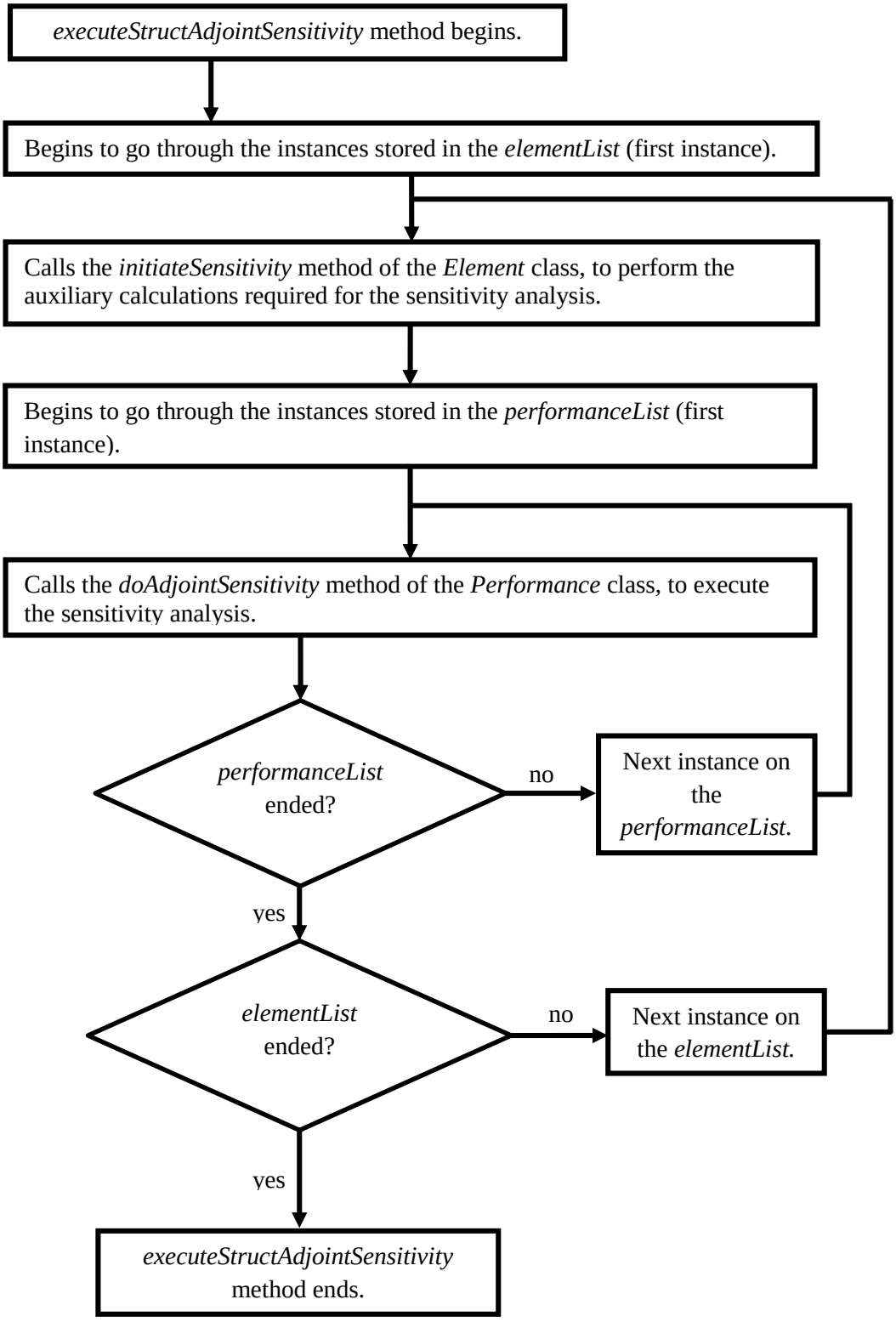

Figure 8.13 – Execution process diagram of the *executeStructAdjointSensitivity* method.

8.5.2. SolutionManager Class

The *SolutionManager* and its derived classes are responsible for finding the solution for the structural analysis. From the three derived classes illustrated in the figure 8.14, only the *ModalAnalysis* class will not be discussed this dissertation.

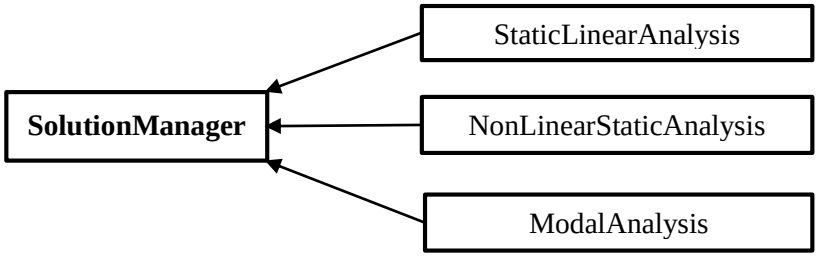

Figure 8.14 – *SolutionManager* derived classes.

The relevant members and the methods of the *SolutionManager* class are presented in the tables 8.5 and 8.6, respectively.

Table 8.5 - Members of the *SolutionManage*r class.

SolutionManager's Members	**Description**
Vector *loadVector*	Stores the assembled structural loads.
LinearEquation * *linearEq*	Pointer for a *LinearEquation* derived class instance.
Model * *model*	Pointer for a *Model* class instance.

Table 8.6 - Methods of the *SolutionManager* class.

SolutionManager*'s* Methods	**Description**
void *doNormalLoadAssemble*(void)	Proceeds to a usual load assembly into the *loadVector* member.
void *setLoad*(int **dofId**, double **value**)	Sets the **value** parameter in the **dofId** position of the *loadVector* member; is used for a displacement performance structural adjoint analysis.
void *doAdjntLoadAssemble*(int **dofId**)	Proceeds to an assembly of the adoint loads of the element with the **dofID** identification into the *loadVector* member; is used for a stress performance structural adjoint analysis.
virtual bool *computeAnalysis*(void)	Initiates the structural analysis.

The *computeAnalysis* virtual method is shared by all the *SolutionManager* derived classes and initiates the analysis that each class is responsible for. The *StaticLinearAnalysis* class is responsible for finding the solution of linear structural problems and also for the adjoint analysis, its specific and relevant methods can be found in the table 8.7.

Table 8.7 - Methods of the *StaticLinearAnalysis* class.

StaticLinearAnalysis*'s* Methods	**Description**
StaticLinearAnalysis(*Model* * **model**)	Constructor; receives an instance of the *Model* class.
StaticLinearAnalysis(*Model* * **model**, int **type**, int **performanceID**, int **dofID**)	Constructor in case of a structural adjoint analysis; the **type** parameter identifies the type of performance; the **performanceID** identifies the performance; the **dofID** parameter can be: a single degree of freedom in case of a displacement performance or the identification of an element in case of a stress performance.
bool *writeDisplacements*(*Vector* & **displacementVector**)	Saves the results of a structural analysis; the **displacementVector** parameter stores the displacement field obtained.
bool *writeAdjointDisplacements*(Vector & **displacementVector**)	Saves the results of a structural adjoint analysis; the **displacementVector** parameter stores the adjoint displacement field obtained.

The *StaticLinearAnalysis* version of the *computeAnalysis* virtual method is shown in figure 8.15.The *StaticNonLinearAnalysis* class has only two methods: its own constructor and its own version of the *computeAnalysis* virtual method shown in figure 8.16.

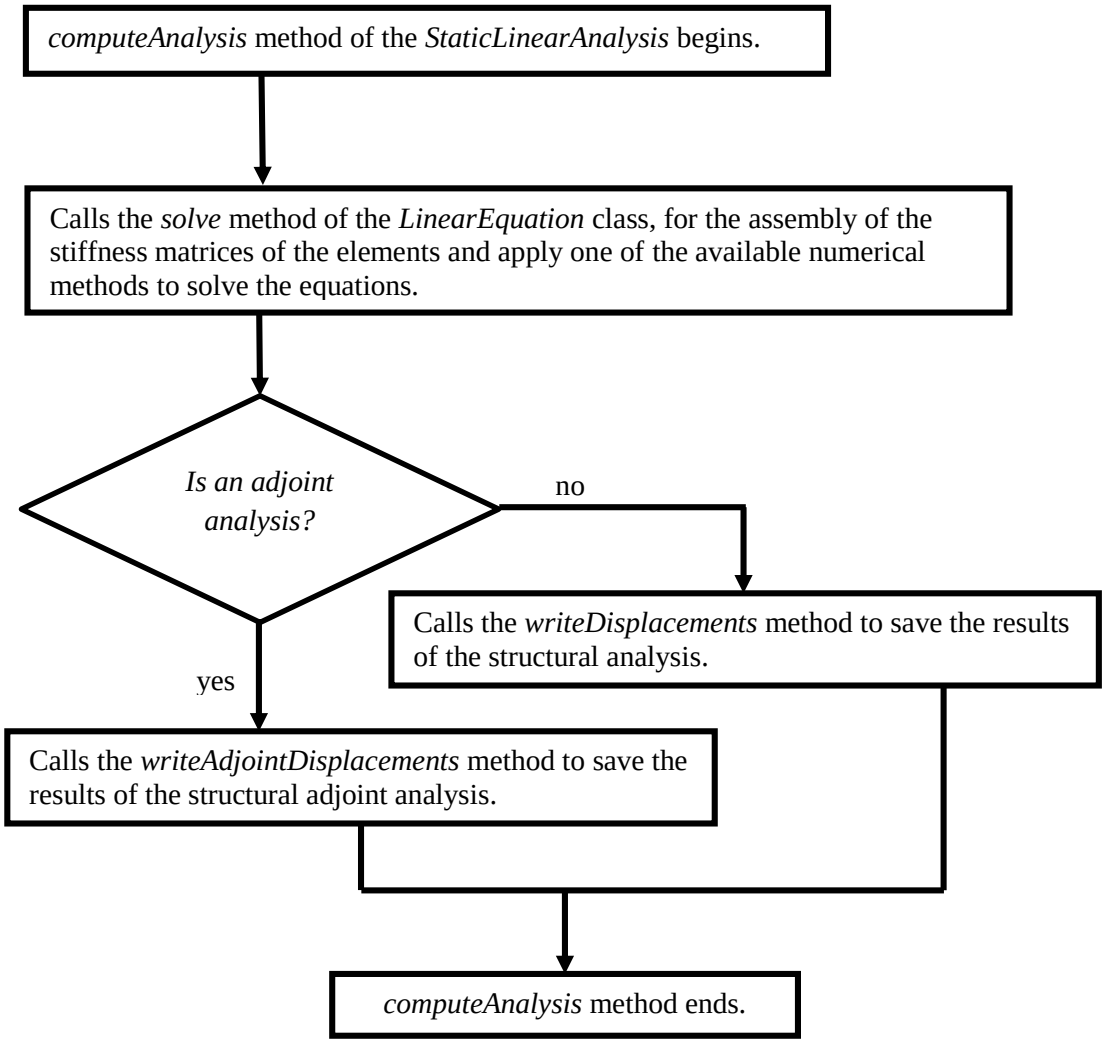

Figure 8.15 - Execution process diagram of the *computeAnalysis* method of the *StaticLinearAnalysis* class.

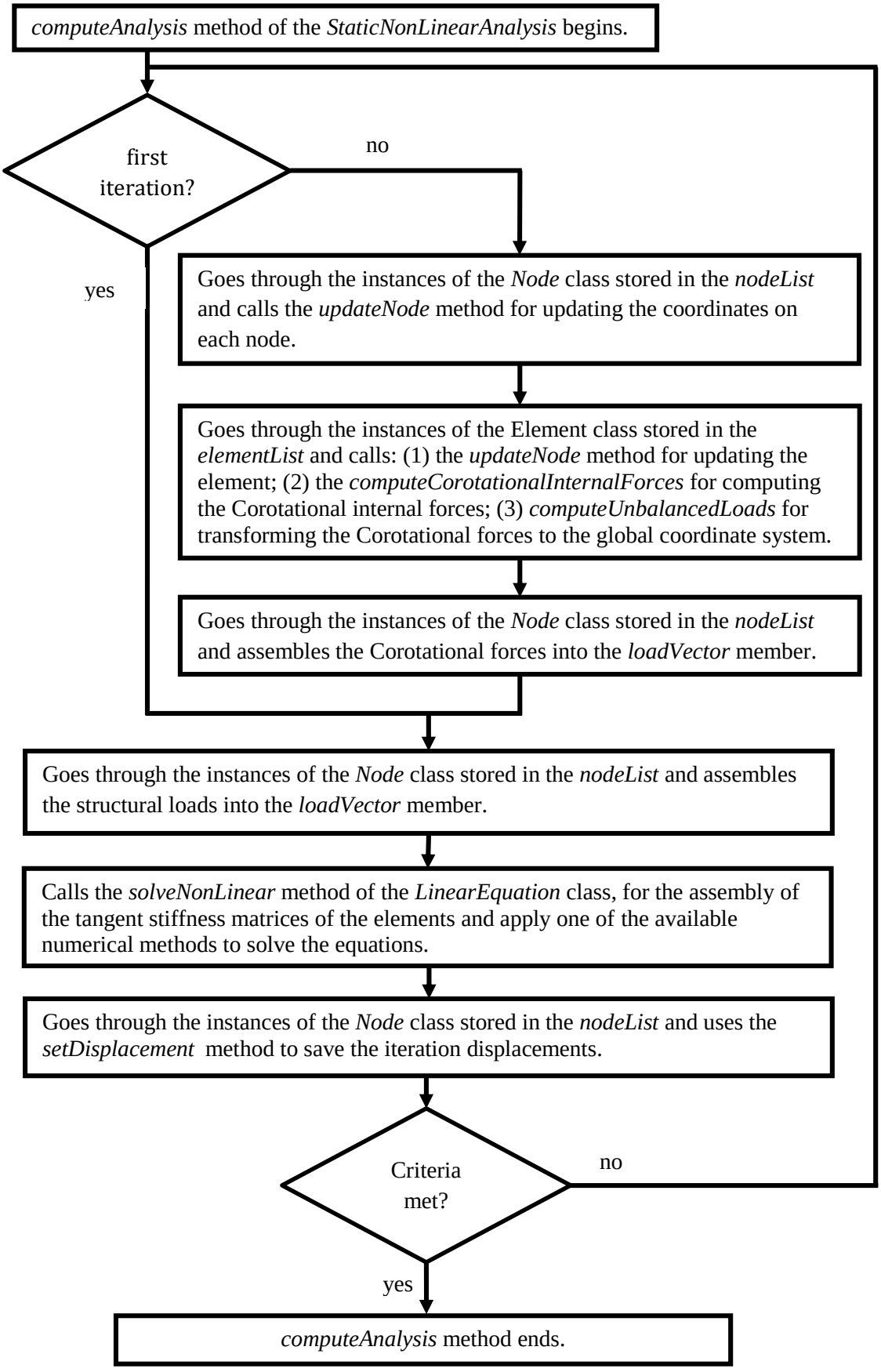

Figure 8.16 - Execution process diagram of the *computeAnalysis* method of the *NonLinearStaticAnalysis* class.

8.5.3. NumericalMethods Class

The *NumericalMethods* class has two derived classes as shown in the figure 8.17.The *Eigen* class has the implementation of the Jacobi numerical method for solving modal problems, which will not be described here.

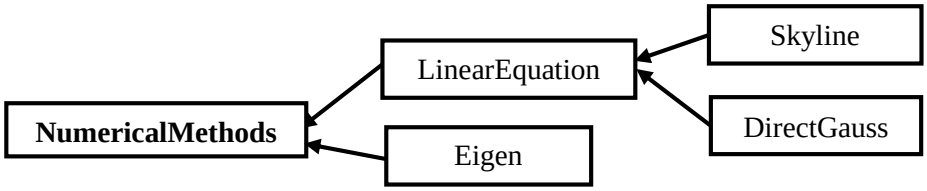

Figure 8.17 - *NumericalMethods* derived classes.

The derived classes of the *LinearEquation* are responsible for the numerical methods needed to solve a system of linear equations. In table 8.8 the *LinearEquation's* methods are presented.

Table 8.8 - Methods of the *LinearEquation* class.

***LinearEquation*'s Methods**	**Description**
LinearEquation(void)	Constructor.
bool *solve*(*Model* * **model**, *Vector* & **b**, *Vector* & **answer**)	Calls the *assembleAndSolve* virtual method; the **b** parameter holds the load vector and the **answer** is where the results are stored.
bool *solveNonLinear*(*Model* * **model**, *Vector* & **b**, *Vector* & **answer**)	Calls the *assembleAndSolveNonLinear* virtual method*;* the **b** parameter holds the load vector and the **answer** is where the results are stored.
virtual bool *assembleAndSolve*(*Model* * **model**, *Vector* & **b**, *Vector* & **answer**)	Assembles the linear stiffness matrices of all the elements and calls a numerical method to solve the equations*;* the **b** parameter holds the load vector and the **answer** is where the results are stored.
virual bool *assembleAndSolveNonLinear*(*Model* * **model**, *Vector* & **b**, *Vector* & **answer**)	Assembles the nonlinear tangent stiffness matrices of all the elements and calls a numerical method to solve the equations*;* the **b** parameter holds the load vector and the **answer** is where the results are stored.

The *assembleAndSolve* and the *assembleAndSolveNonLinea*r virtual methods will be shown again in the methods tables of the *Skyline* and *DirectGauss* classes, since they call different methods during their execution. The *DirectGauss* class employs the normal Gaussian elimination algorithm for solving systems of equations and its methods are shown in the table 8.9.

Table 8. 9 - Methods of the *DirectGauss* class.

***DirectGauss'*s Methods**	**Description**
DirectGauss(void)	Constructor.
bool *assembleAndSolve*(*Model* * **model**, *Vector* & **b**, *Vector* & **answer**)	Calls the *computeStiffnessMatrix* method of each instance of the *Element* class stored in the *elementList*, to calculate the stiffness matrix, which is then assembled in the global stiffness matrix. Finally calls the *solveForRhs* method.
bool *assembleAndSolveNonLinear*(*Model* * **model**, *Vector* & **b**, *Vector* & **answer**)	Calls the *computeNonLinearStiffnessMatrix* method of each instance of the *Element* class stored in the *elementList*, to calculate the tangent stiffness matrix, which is then assembled in the global stiffness matrix. Finally calls the *solveForRhs* method.
bool *solveForRhs*(*Vector* & **b**, *Vector* & **answer**, *Matrix* & **K**);	Employs the Gaussian elimination method; the K parameter is the global stiffness matrix, the **b** parameter holds the load vector and the **answer** is where the results are stored.

The *Skyline* class employs a more efficient numerical method for larger and more complex problems using a factorization algorithm, where the global stiffness matrix is decomposed into a LDL^T factorization, where L is the lower triangular and D is the diagonal of the global stiffness matrix . The skyline storage method only requires two one-dimensional arrays, which are actually the *Skyline* class members: (1) the *mtrx* array member for the values of the coefficients of the global stiffness matrix; (2) the *adr* Vector for the addresses of the diagonal coefficients of the stiffness matrix on the *mtrx* array.

Table 8.10 - Methods of the *Skyline* class.

***Skyline'*s Methods**	**Description**
Skyline(void)	Constructor; calls the *preSkyline* method.
void *preSkyline*(*Model* * **model**)	Ascertains the dimension of the coefficient array and pre allocates the space into *mtrx* array member.
bool *assembleAndSolve*(*Model* * **model**, *Vector* & **b**, *Vector* & **answer**)	Calls the *computeStiffnessMatrix* method of each instance of the *Element* class stored in the *elementList*, to calculate the stiffness matrix, which is then assembled in the *mtrx* member array, though the *assemble* method. Finally calls the *backSubstitutionWith* method.
bool *assembleAndSolveNonLinear*(*Model* * **model**, *Vector* & **b**, *Vector* & **answer**)	Calls the *computeNonLinearStiffnessMatrix* method of each instance of the *Element* class stored in the *elementList*, to calculate the stiffness matrix, which is then assembled in the *mtrx* member array, though the *assemble* method. Finally calls the *backSubstitutionWith* method.
bool *assemble*(*Matrix* * **kmatrix**, int * **dofs**)	Assembles the stiffness matrix of one element into the *mtrx* member; the **Kmatix** parameter is the stiffness of the element and the **dofs** parameter is the degrees of freedom of the element.
bool *backSubstitutionWith*(*Vector* & **b**, *Vector* & **answer**);	Employs the backward substitution method; the **b** parameter holds the load vector and the **answer** is where the results are stored.

8.5.4. StructuralComponents Class

The *StructuralComponents* and its derived classes are undoubtedly one of the most extensive group, as they contain all necessary objects to represent a finite element model, as illustrated in figure 8.18.

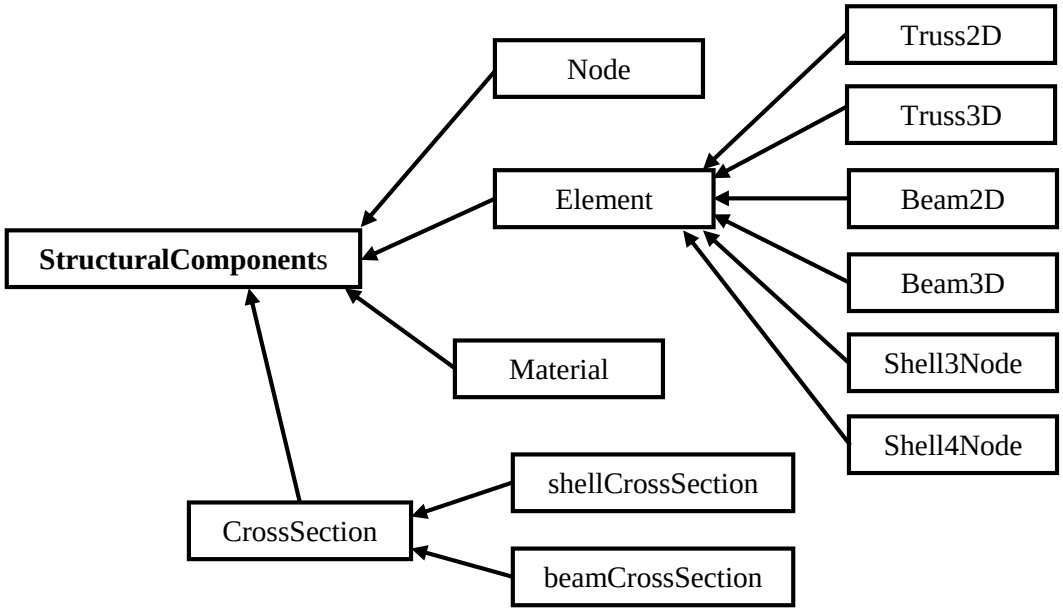

Figure 8.18 – *StructuralComponents* derived classes.

The *Node* class stores information regarding the nodes of the mesh. Considering that the nodes are the element's boundaries and the connection between them, the information they hold is quite extensive. Similarly to the *Model* class, *Node* contains three lists, as shown in Fig. 8.19 and table 8.11.The *Node* class has a total of 29 members and 47 methods.

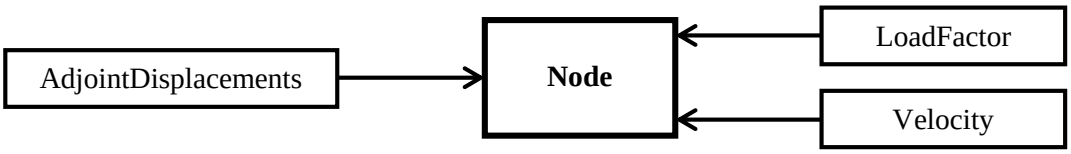

Figure 8.19 - Classes which are contained in *Node* lists.

Table 8.11 - Lists of the *Node* class.

***Node's* Lists**	**Description**
std::list <Node *> *nodeList*	Stores all the objects of the *Node* class type.
std::list < *Performance* *> *adjntDisplacementList*	Stores all the objects of the *AdjointDisplacements* class type.
std::list <*Node* *> *velocityList*	Stores all the objects of the *Velocity* class type.

In the tables 8.12 and 8.13 the more relevant members of the *Node* class are shown, and in the tables 8.14 and 8.15 the more important methods are presented.

Table 8.12 - Members for the *Node* class.

Node's **Members**	**Description**
Int *ID*	Stores the identification of the instance.
double *x*	Stores the value of the x-coordinate of the node.
double *y*	Stores the value of the y-coordinate of the node.
double *z*	Stores the value of the z-coordinate of the node.
double *initialCoordX*	Stores the initial value of the x-coordinate of the node.
double *initialCoordY*	Stores the initial value of the y-coordinate of the node.
double *initialCoordZ*	Stores the initial value of the z-coordinate of the node.
int *dof*	Stores the number of degrees of freedom of the node.
int *perfDofID*	Stores the identification of the degrees of freedom, which was declared as a performance.
double *loadValue*[6]	Stores the values of the loads applied to the structure, in the following order: *loadValue*[0] – value of the force in the x-direction; *loadValue*[1] – value of the force in the y-direction; *loadValue*[2] – value of the force in the z-direction; *loadValue*[3] – value of the moment about the x-axis; *loadValue*[4] – value of the moment about the y-axis; *loadValue*[5] – value of the moment about the z-axis.
bool *activeDofs*[6]	Identifies which degrees of freedom is active in the node: *activeDofs*[0] – for the x-direction displacement (U_x); *activeDofs*[1] – for the y-direction displacement (U_y); *activeDofs*[2] – for the z-direction displacement (U_z); *activeDofs*[3] – for the rotation in turn of the x-axis (Θ_x); *activeDofs*[4] – for the rotation in turn of the y-axis (Θ_y); *activeDofs*[5] – for the rotation in turn of the z-axis (Θ_z).

Table 8.13 - Continuation of the members for the *Node* class.

***Node's* Members**	**Description**
bool *impDisplacement*[6]	Identifies which degrees of freedom have initial restrictions imposed: *impDisplacement*[1] – restrictions on the U_y degree of freedom; *impDisplacement*[2] – restrictions on the U_z degree of freedom; *impDisplacement*[3] – restrictions on the Θ_x degree of freedom; *impDisplacement*[4] – restrictions on the Θ_y degree of freedom; *impDisplacement*[5] – restrictions on the Θ_z degree of freedom.
int *NodalDofID*[6]	Stores the identification of the degrees of freedom of the node: *NodalDofID*[0] – ID of the U_x degree of freedom; *NodalDofID*[1] – ID of the U_y degree of freedom; *NodalDofID*[2] – ID of the U_z degree of freedom; *NodalDofID*[3] – ID of the Θ_x degree of freedom; *NodalDofID*[4] – ID of the Θ_y degree of freedom; *NodalDofID*[5] – ID of the Θ_z degree of freedom.
double *unbalancedLoads*[6]	Stores the values of the internal Corotational forces: *unbalancedLoads*[0] – value of the force in the x-direction; *unbalancedLoads*[1] – value of the force in the y-direction; *unbalancedLoads*[2] – value of the force in the z-direction; *unbalancedLoads*[3] – value of the moment about the x-axis; *unbalancedLoads*[4] – value of the moment about the y-axis; *unbalancedLoads*[5] – value of the moment about the z-axis.
bool *DV*	Identifies if the node has a coordinate which is a design variable.
Bool *hasImpDisplacement*	Identifies if the node has a restrictions imposed.
Bool *hasLoad*	Identifies if the node has structural loads applied.
Vector *displacements*	Stores the displacement field of the node, obtained in the structural analysis.

Table 8.14 - Methods for the *Node* class.

Node's **Methods**	**Description**
Node(int **ID**, double **x**, double **y**,double **z**)	Constructor; receives the identification and the node coordinates as parameters.
void *updateNode*(void)	Updates the node coordinates.
int *getNodeID*(void)	Returns the *ID* member.
int *getDof*(void)	Returns the *dof* member.
int *setDisplacement*(const char * **label**, double **value**)	Sets the **value** parameter, into the position defined in **label** parameter, on the *displacement* member.
Vector *getDisplacemen*t(void)	Returns the *displacement* member.
bool *hasImpDisplacements*(void)	Returns the *hasImpDisplacement* member.
bool *setImpDisplacement*(const char * **str1**, double **value**);	Sets the **value** parameter, into the position defined in **label** parameter, on the *impDisplacement* member.
bool * *getImpDisplacement*(void)	Returns the *impDisplacement* member.
bool *setUnbalancedLoad*(const char * **label**, double **load**)	Sets the **load** parameter, into the position defined in **label** parameter, on the *unbalancedLoads* member.
double * *getUnbalancedLoads*(void)	Returns the *unbalancedLoads* member.
double *getInitialCordX*	Returns the *initialCoordX* member.
double *getInitialCordY*	Returns the *initialCoordX* member.
double *getInitialCordZ*	Returns the *initialCoordX* member.
void *setNodalDofID*(int **dof**, int **id**)	Sets the **dof** parameter into the **id** position of the *NodalDofID* member.
int *getNodalDofID*(int **dof**)	Returns the identification into the **dof** position of the *NodalDofID* member.
int * *getNodalDofID*(void)	Returns the *NodalDofID* member.
void *activeDof*(int **dof**)	Sets **true** into the **dof** position of the *activeDofs* member.
bool * *getActiveDofs*(void)	Returns the *activeDofs* member.
void *setLoads*(const char * **label**, double **load**)	Sets the **load** parameter, into the position defined in **label** parameter, on the *loadValue* member.

Table 8.15 - Continuation of the methods for the *Node* class.

***Node's* Methods**	**Description**
void *setLoads*(const char * **label**, double **load**)	Sets the **load** parameter, into the position defined in **label** parameter, on the *loadValue* member.
bool *hasLoads*(void)	Returns the *hasLoad* member.
void *setNewAdjntDisplacementIt* (int **ID**)	Creates a new instance of the *AdjointDisplacement* class with the **ID** identification and stores it in the *adjntDisplacementList* member.
std::list <AdjointDisplacement *> *getAjntDispList*(void)	Returns the *adjntDisplacementList* member.
std::list < Velocity *> *getVelocityList*(void)	Returns the *velocityList* member.
int *getPerfNodeID*(int **perfDofID**)	Returns the degree of freedom into the (**perfDofID**) position of the *NodalDofID* member.

The *CrossSection* class holds all the parameters relating the cross section of the elements and has two derived classes: (1) *ShellCrossSection* class; (2) *BeamCrossSection* class. The methods of the *CrossSection* class are just for accessing its members, and are shown in the table 8.16.

Table 8.16 - Members of the *CrossSection* class.

***CrossSection's* Members**	**Description**
double *Iy*	Stores the value of the inertia relative to the y-axis.
double *Iz*	Stores the value of the inertia relative to the z-axis.
double *A*	Stores the value of the Cross Section area.
double *h*	Stores the value of the Cross Section thickness.
int *numOfVariables*	Stores the number of design variables in this instance.
bool *hasADV*	Identifies if the area was declared a design variable.
bool *hasIyDV*	Identifies if the *Iy* was declared a design variable.
bool *hasADV*	Identifies if the *Iz* was declared a design variable.
int *IyVariableID*	Stores the identification of the area design variable.
int *IzVariableID*	Stores the identification of the *Iy* design variable.
int *AVariableID*	Stores the identification of the *Iz* design variable.

The Material class holds all the parameters related to the material properties, such as in the *CrossSection* class, its methods are just intended for accessing to its members, so it is not worth showing them. The members of the *Material* class are presented in the table 8.17.

Table 8.17 - Members of the Material class.

Material's **Members**	**Description**
double *e*	Stores the Young's modulus value.
double *r*	Stores the density value.
double *v*	Stores the Poisson coefficient value.
int *variableID*	Stores the identification of the design variable.
bool *hasDV*	Identifies if it was declared a design variable.

The *Element* class has 6 derived classes, each one representing an element that has been introduced in the previous chapters. The *Element* class also has a list for its gauss points, which is shown with other relevant members in the tables 8.18 and 8.19. In the tables 8.20 and 8.21 the more important methods are presented.

Table 8.18 - Members of the Element class.

Element's **members**	**Description**
std::list < *GaussPoint* *> *gaussPoint*	Stores all the objects of the *GaussPoint* class type.
int *dof*	Stores the number of degree of freedom of the element.
int *numberOfNode*	Stores the number of nodes of the element.
std::list <CrossSection *>::iterator *crossSection*	Iterator for an instance of the *CrossSection* class stored in the *Model's crossSectionList* list.
std::list <Material *>::iterator *material*	Iterator for an instance of the *Material* class stored in the Model's *materialList* list.
std::list <Node * >::iterator *node1, node2, node3, node4*	Iterators for an instance of the *Node* class stored in the *Model's nodeList* list.
int *type*	Stores the ID of the class type in the instance: 1 –Truss2D class; 2 –Truss3D class; 3 –Beam2D class; 4 –Beam3D class; 5 –Shell3Node class; 6 –Shell4Node class.

Table 8.19 - Continuation of the members of the *Element* class.

Element's members	Description
double *length*	Stores the value of the length of the element.
double *initialLength*	Stores the value of the initial length of the element.
Vector * *internalForces*	Stores the values of the internal forces of the element.
Vector *elementDisp*	Stores the values of the displacements of the nodes of the element, obtained in the structural analysis.
int * *elementDofID*	Stores the degrees of freedom of the nodes of the element.
int *numberOfGauss*	Stores the number of gauss necessary for the element.
int *numOfPerformances*	Stores the *number of* performances declared on the element.
Matrix *shapeFunctionStateGradRot*	Stores the gradient shape functions coefficients for the rotations degrees of freedom.
Matrix *shapeFunctionStateGrad*	Stores the gradient shape functions coefficient for the rotations degrees of freedom.
Matrix jacobianMatrix	Stores the Jacobian Matrix.
Matrix *strainDispMatrix*	Stores the Strain-Displacement Matrix.

Table 8.20 - Methods of the *Element* class.

Element's methods	Description
int *getDof*(void)	Returns the *dof member*.
int *getType*(void)	Returns the *type* member.
double *getLength*(void)	Returns the *length* member.
Vector & *getInternalForces*(void)	Returns *elementDisp* member.
int *getNode1ID*(void)	Returns the identification of the instance of the *Node* class, referenced by the *node1* iterator member.
int *getNode2ID*(void)	Returns the identification of the instance of the *Node* class, referenced by the *node2* iterator member.
int *getNode3ID*(void)	Returns the identification of the instance of the *Node* class, referenced by the *node3* iterator member.

Table 8.21 - Continuation of the methods of the *Element* class.

Element's methods	Description
int *getNode4ID*(void)	Returns the identification of the instance of the *Node* class, referenced by the *node4* iterator member.
int *getMaterialID*(void)	Returns the identification of the instance of the *Material* class, referenced by the *material* iterator member
int *getCrossSectionID*(void)	Returns the identification of the instance of the *CrossSection* class, referenced by the *crossSection* iterator member.
void *computeLength*(void)	Computes the length of the element.
void *updateElement*(void)	Updates the length of the element.
Matrix * *computeStiffnessMatrix*(void)	Calls the *computeLocalStiffnessMatrix* virtual method to obtain the element stiffness matrix in the local coordinate system, then through the rotation matrix obtained from the *computeRotationMatrix* virtual method, the stiffness matrix is transformed to the global coordinate system.
Matrix * *computeNonLinearStiffnessMatrix*(void)	Calls the *computeLocalStiffnessMatrix* and the *computeLocalGeometricMatrix* virtual methods to calculate the nonlinear tangent stiffness matrix, then through the rotation matrix obtained from the *computeRotationMatrix* virtual method, the tangent stiffness matrix is transformed to the global coordinate system.
void *computeInternalForces*(void)	Computes the internal forces of the element, by using the rotation matrix obtained from the *computeRotationMatrix* method, the local stiffness matrix obtained from the *computeLocalStiffnessMatrix* method and the displacements in the *elementDisp member*.

The methods of the *Element* class that are not show here only request different types of information to the instances of the classes stored in the iterators. It is however relevant to present the *Element* virtual methods, which are methods that are defined in all the derived classes, although the code in each version may be different.

Some of these virtual methods actually haven't been completely implemented in all the derived classes, for example, the methods related do the nonlinear analysis only have been implemented in *Beam2D*. This is a main feature of object oriented languages, such as C++, and allows establishing a framework for future software development. Once the global architecture is

defined, all relevant classes and virtual methods are created. Then, each derived class is necessarily created with the proper virtual methods defined, but with the code for these methods eventually left blank. The code for these methods can be added later, and the built-up of the functionalities that were planed left for future development. The virtual methods of the *Element* class are presented in the table 8.22, it is also indicated, in the last column of the table, in which derived classes those methods were implemented on. The *Truss2D* and *Beam2D* classes contain all the necessary auxiliary methods to perform the analysis of sensitivities, which are shown in the table 8.24.

Table 8.22 - Virtual methods of the Element class.

Element's methods	**Description**	**Derived Classes implementation**
virtual void *setElementDofID*(void)	Sets the *elementDofID* member.	All
virtual void *computeCorotationalInternalForces*(void)	Computes the Corotational internal forces and stores them into the *internalForces* member.	*Beam2D* and *Beam3D* classes
virtual void *computeUnbalancedLoads*(void)	Transforms the local Corotational internal forces to the global coordinate system through the rotation matrix obtained from the *computeRotationMatrix* method, and then calls the *setUnbalancedLoad* of the instances of the *Node* class, referenced by the *node1* and *node2* members, to set the forces in the nodes.	*Beam2D* and *Beam3D* classes
virtual Matrix & *computeLocalStiffnessMatrix*(void)	Computes the Stiffness matrix of the element in the local coordinate system.	All
virtual Matrix & *computeLocalGeometricMatrix*(void)	Computes the geometric matrix of the element in the local coordinate system.	*Beam2D* and *Beam3D* classes
virtual Matrix & *computeRotationMatrix* (double **l**, double **dx**, double **dy**, double **dz**)	Computes the rotation matrix of the element, using the values passed as parameters (the length and the direction cosines) for the calculations.	All
virtual void *initiateSensitivity*(int **numOfPerformances**)	Initiates all the auxiliary calculations required for the sensitivity analysis, it calls the computeStrainDispMatrix, the computeOrigGaussData, the computeAdjntGaussData and the *computeVelocity* methods, which are shown in the table 8.24.	*Truss2D* and *Beam2D* classes

Table 8.23 - Virtual methods of the *Element* class.

Element's methods	Description	Derived Classes implementation
virtual Vector & *doAPrime*(int **ID**)	Calculates the sensitivities analytic integral expressions, using the information stored in the instances of the *GaussPoint* class, in the *gaussPointList* list.	*Truss2D* and *Beam2D* classes
virtual Vector & *doVonMisesPrime*(int **ID**, int **point**)	Calculates the sensitivities analytic integral expressions, of the stress performance, using the information stored in the instances of the *GaussPoint* class, in the *gaussPointList* list.	*Truss2D* and *Beam2D* classes
virtual Vector & *doVolumePrime*(int **ID**)	Calculates the sensitivities analytic integral expressions, of the volume performance, using the information stored in the instances of the *GaussPoint* class, in the *gaussPointList* list.	*Truss2D* and *Beam2D* classes

Table 8.24 - Auxiliary methods for sensitivity analysis of the Truss2D and Beam2D classes.

Methods of the Truss2D and Beam2D classes	Description
void *computeStrainDispMatrix* (void)	Computes the Strain-Displacement Matrix and stores it in the *strainDispMatrix* member.
void *computeJacobianMatrix* (void)	Computes the Jacobian Matrix and stores it in the *jacobianMatrix* member.
void *computeOrigGaussData* (void)	Begins by requesting the original displacements, which are used to compute all the necessary data for the Gauss points from the original structural analysis, is then created a new instance of the *GaussPoint* class, which is stored in the *gaussPointLis*t list.
void *computeAdjntGaussData* (void)	Begins by requesting the lists with the adjoint displacements (*adjntDisplacementList*) from the *Node* class iterators, which are used to compute all the necessary adjoint data for the Gauss points, then goes through the instances of the *GassPoint* class stored in the *gaussPointList* list, created in the *computeOrigGaussData method,* and adds the new adjoint data.
void *computeVelocity*(void)	Computes the velocity field, then goes through the instances of the *GassPoint* class stored in the *gaussPointList* list, created in the *computeOrigGaussData method,* and adds the velocity data.

8.5.5. Sensitivity Class

The *Sensitivity* and its derived classes are only used when performing a design sensitivity analysis, being one of the main reasons why they have been grouped together.

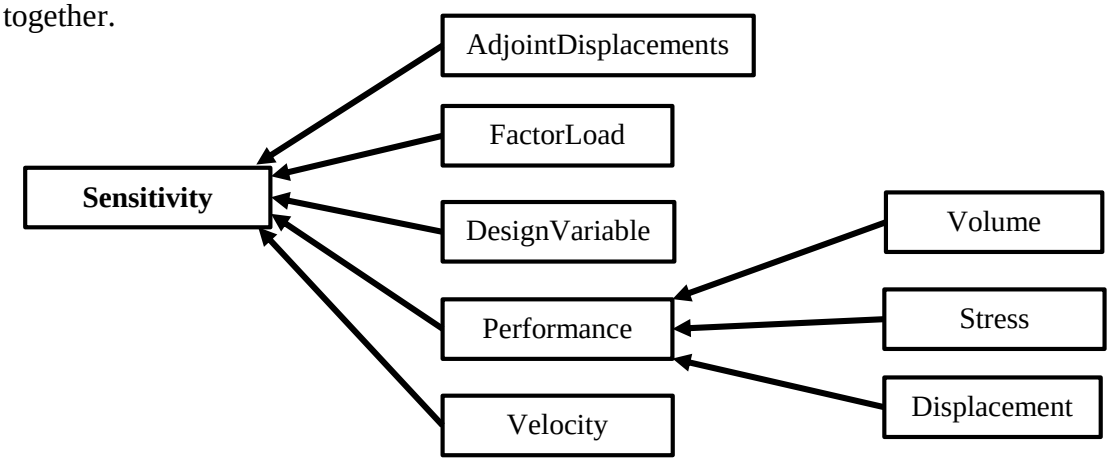

Figure 8.20 - Sensitivity and its derived classes.

From the classes shown in figure 8.20, only the *Performance* and its derived classes execute different tasks besides storing information. The *AdjointDisplacements* class stores the information relating to the adjoint displacements, one instance of this class is created for each instance of the *Node* class defined and each new adjoint analysis. The *FactorLoad* class stores the value of a particular load factor. The *Velocity* class stores the information of the calculated velocity field. The instances of all the above classes are stored in lists in the *Node* class. The *DesignVariable* class saves instances of a class that has been declared as a design variable, which can be:

- *Node* class – The design variable value can be one of the coordinates of the node;
- *CrossSection* class – The design variable value can be the area or the inertia of the cross section;
- *Material* class – The design variable value is the Young's coefficient;
- *LoadFactor* class – The design variable value is the load factor.

In fact, the class does not properly stores an instance of a class, but a iterator for the instance of the class in the *Model*'s list where the instance is stored, so that, when in the reading process, the *ReadOptFile* class has direct access to the instance of the class being updated and the changes to the values of that class are then general and definitive.

The *Performance* class has the responsibility to redirect all sensitivities and performances tasks to one of the three derived classes. The relevant members and virtual methods of the performance and its derived classes are described in the tables 8.25 and 8.26, respectively. The only methods that have a different name for each of the derived classes are in fact their own constructors. Table 8.27 shows the constructors of the three derived classes.

The *DesignVariable* members and methods are presented in the tables 8.28 and 8.29, respectively.

Table 8.25 - Members of the *Performance* class.

Performance's members	Description
double *performanceValue*	Stores the value of the performance.
int *numberOfDV*	Stores the number of total design variable defined.
Vector *sensitivityVector*	Stores the results of the sensitivity analysis, the performance gradients.
std::list <Node *>::iterator *node*	Iterator for an instance of the Node class stored in the Model's *nodeList* list.
std::list <Element *>::iterator *element*	Iterator for an instance of the Element class stored in the Model's *elementList* list.
coordinateDir	Sores the direction of the displacement in case of the performance being of the displacement: 1 – X-direction; 2 – Y-direction.

Table 8.26 - Virtual methods of the *Performance* class.

Performance's virtual methods	Description
virtual int *doAdjointStructAnalysis (void)*	Sets the adjoint load and identifies the type of the derived class of the Performance base, returns: 1- If is an instance of the *Volume* class; 2- If is an instance of the *Displacement* class; 3- If is an instance of the *Stress* class.
Virtual void *setPerformanceValue* (void)	Calculates the value of the performance.
virtual int *doAdjointStructAnalysis* (void)	Initiates the sensitivity calculations and stores the results in the *sensitivityVector* member.
int *getPerfNodeID*(void)	Returns the degree of freedom of the displacement defined as a performance.
Virtual void *doAdjointSensitivity* (std::list<Element *> :: iterator **elemIt**, int **numberOfDV**)	Sends the instances of the *Element* class to the instances of the *Performance* class, to perform the sensitivity analysis.

Table 8.27 - Constructors of the derived classes of *Performance.*

Constructors	**Description**
Displacement(int **ID**, std::list <Node *>::iterator **noIt**, int **CoordinateDir**)	Constructor of the *Displacement* Class. Sets the *node* iterator, the *ID* and the *coordinateDir* members with the **noIt, ID** and **CoordinateDir** parameters, respectively.
Stress(int **ID**, std::list < Element *>::iterator **elemIt**, int **point,** double **pos**)	Constructor of the *Stress* Class. Sets the *croSec* iterator member with the *crosIt* iterator received as a parameter and the variable member with the parameter type.
Volume(int **ID**)	Constructor of the *Volume* Class. Receives the identification of the instance as a parameter

Table 8.28 - Members of the *DesignVariable* class.

DesignVariable's **members**	**Description**
std :: list < LoadFactor * > :: iterator *loadFactor*	Iterator for an instance of the *LoadFactor* class stored in the Node's *loadFactorList* list.
std :: list < CrossSection * > :: iterator *croSec*	Iterator for an instance of the *CrossSection* class stored in the Model's *crossSectionList* list.
std :: list < Material * > :: iterator *mat*	Iterator for an instance of the *Material* class stored in the Model's *materialList* list.
std ::list < Node * > :: iterator *node*	Iterator for an instance of the *Node* class stored in the Model's *nodeList* list.
int *ID*	Identification of the instance.
int *designVariableType*	Identifies the type of design variable that a particular instance stores: 1 – Young's coefficient (*Material* class); 2 – Inertia (*CrossSection* class); 3 – Area (*CrossSection* class); 4 – Load factor (*LoadFactor* class); 5 – Coordinate (*Node* class);
int *nodeCoord*	Identifies the coordinate: 1 – X-Coordinate; 2 – Y-Coordinate; *3* – Z-Coordinate.

Table 8.29 - Members of the *DesignVariable* class.

***DesignVariable's* methods**	**Description**
DesignVariable(int **ID**)	Constructor; sets the *ID* member with the **ID** parameter.
int *getID*(void)	Returns the *ID* member.
int *getDesignVariableType*(void)	Returns the *designVariableType* member.
void *setDVMat*(std::list<Material *>::iterator **matIt**)	Sets the *mat* iterator member with the **matIt** iterator received as a parameter.
void *setDVCross*(std::list<CrossSection *>::iterator crosIt, int **type**)	Sets the *croSec* and the *variableType* members with the **crosIt** and the **type** parameters, respectively.
void *setDVLoadFactor*(std::list<LoadFactor *>::iterator **loadIt**)	Sets the *loadFactor* iterator member with the **loadIt** iterator received as a parameter.
void *setDVNode*(std::list<Node *>::iterator **nodeIt**, int **N2**)	Sets the *node* and the *nodeCoord* members with the **nodeIt** and the **N2** parameters, respectively.
void *changeDVariableValue*(double **value**)	Access the instance referenced by the iterator stored and updates the design variable value with the **value** parameter.

8.5.6. InOutputManager class

The *InOutputManager* class with its derived classes handles all the operations regarding the input and output of data. This class has 4 derived classes, as seen in figure 8.21:

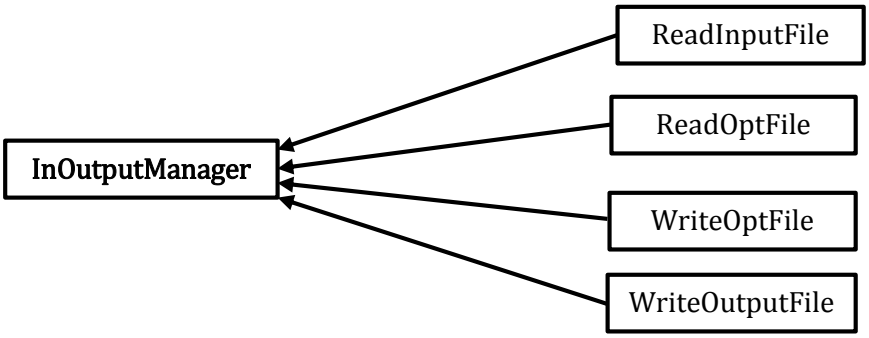

Figure 8.20 - *InOutputManager* derived classes.

The only method of the *InOutputManager* class is its own constructor. Its members are presented in the table 8.30.

Table 8.30 - Members of the *InOutputManager* class.

InOutputManager's **Members**	**Description**
char * *file*	Stores the name of one file handle by the *InOutputManager*.
Model * *model*	Stores an instance of the *Model* class.

The *ReadInputFile* class is responsible for reading the input file regarding the mechanical structure, creating the instances of the classes needed to describe the problem and store them in their respective list in the *Model* class. The 2 methods of the *ReadInputFile* class are shown in the table 8.31.

Table 8.31 - Methods of the *ReadInputFile* class.

ReadInputFile's Methods	**Description**
ReadInputFile(char * inFile, Model * model)	Constructor; receives an instance of the *Model* class and the name of the input data file.
int *readFile* (void)	Initiates the process of reading the input file, the method returns: 0 if the reading was not successful; 1 if the reading was successful; (2) to execute directly the analysis.

The list of commands possible to insert in the input file and be read by the *ReadInputFile* class is described in the EFFECT's user manual, however in table 8.32, 4 examples of possible commands and the respective actions they trigger are presented.

Table 8.32 - Examples of commands for the input data file.

Command	Observations
N	Creates an instance of the *Node* class
MP	Creates an instance of the *Material* class
R	Creates an instance of the *CrossSection* class
ANTYPE	Commands EFFECT to automatically execute a structural analysis, after the input file has been read.

The *ReadOptFile* class is responsible for reading the *opt2effect* file and is only created in the optimization mode of EFFECT, since it is only in the optimization process that the update of the design variables with the values optimized by MATLAB is required. The format of the *opt2effect* file is described in EFFECT's user manual, but an example of the file can be seen in the figure 8.21.

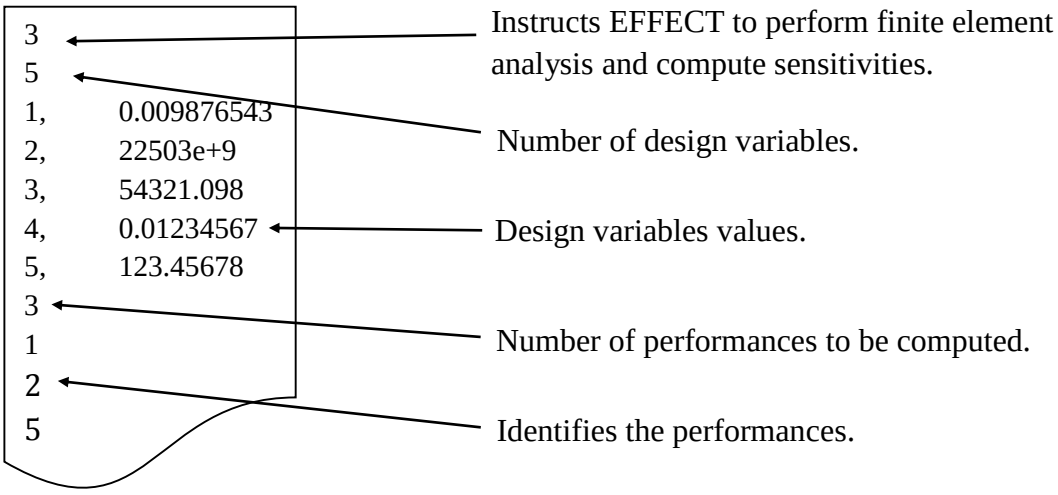

Figure 8.21 - Example of the *opt2effect* file.

The *ReadOptFile* class has only two methods: (1) constructor; (2) *readFile* method. The mechanism that has been created in order to efficiently change the values of design variables combines the *ReadOptFile* class with the *DesignVariable* class as shown in the diagram of *readFile* method in the Figure 8.23.

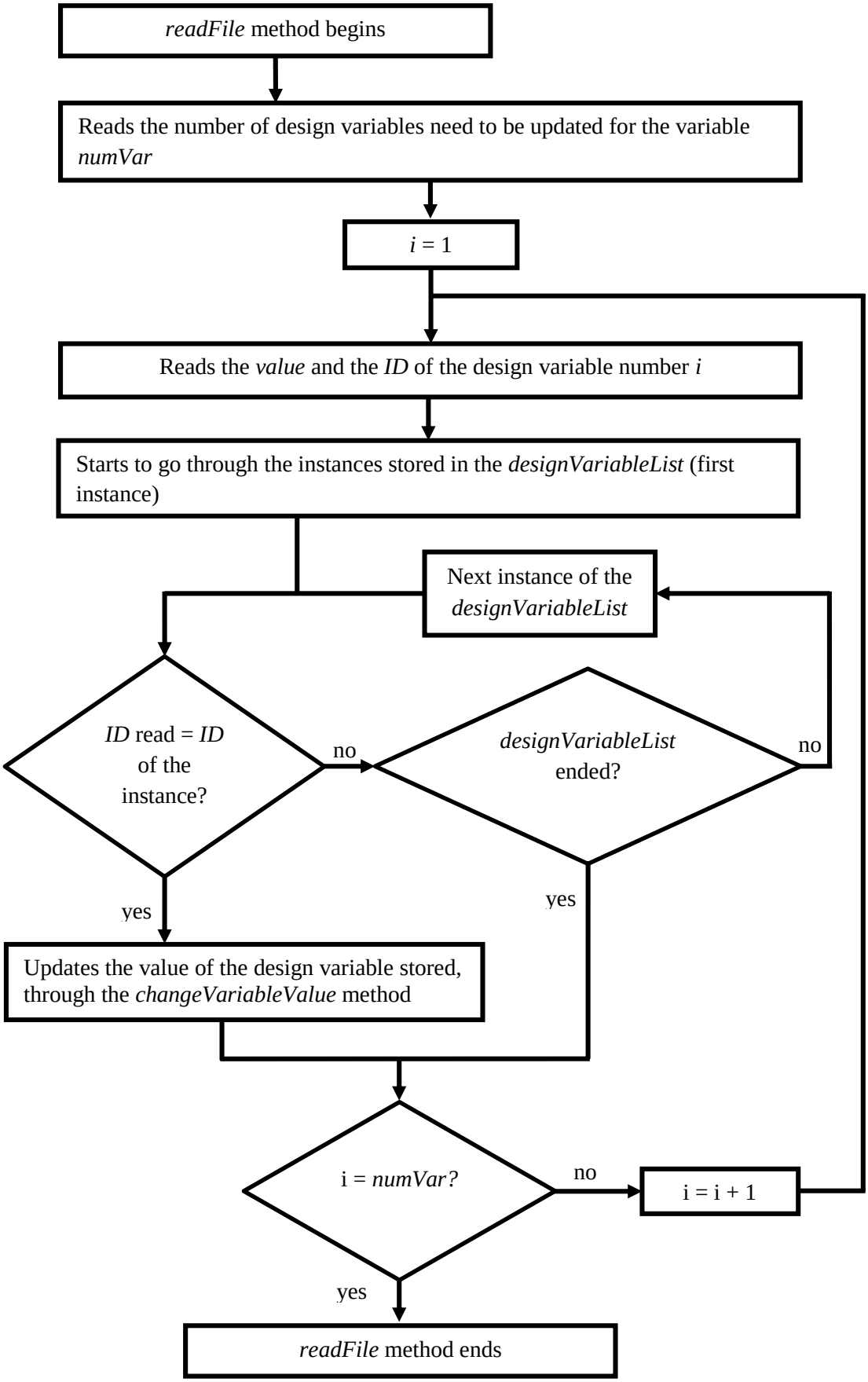

Figure 8.22 – Execution process diagram of the r*eadFile* method of the *ReadOptFile*.

The *WriteOutputFile* class saves the results of the analysis performed into a file and the *WriteOptFile* class writes the results of the structural and the sensitivity analysis in the *efect2opt* for MATLAB optimization. Both classes have only two methods: (1) constructor; (2) *writeFile* method. Once again the format of the *efect2opt* is described in EFFECT's user manual, but an example of the file can be seen in the figure 8.23.

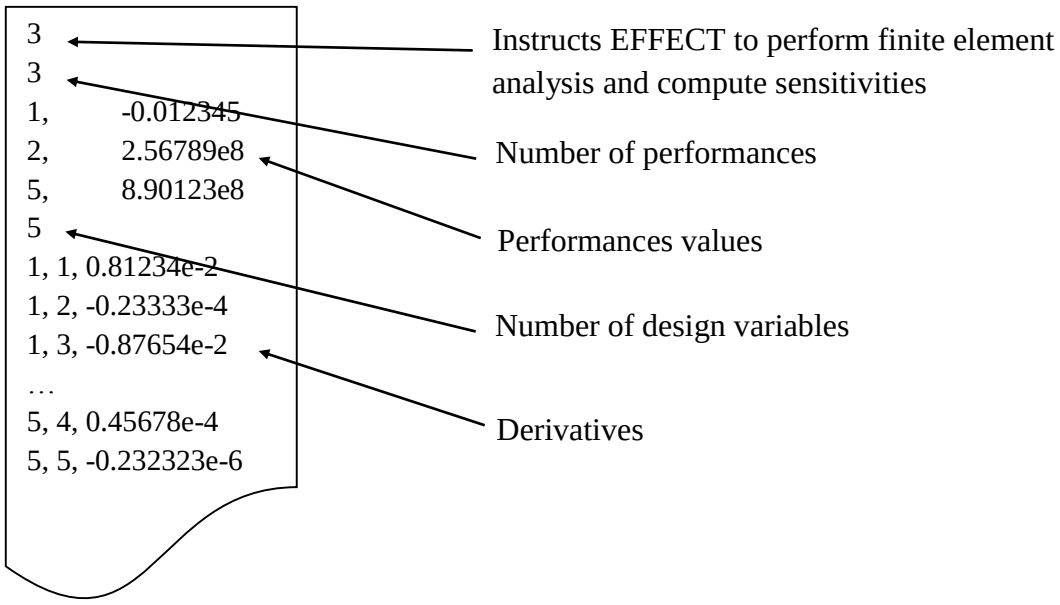

Figure 8.23 - Example of the *effect2opt* file

Chapter 9

9. Test Examples and Verification of Results

9.1. Overview

This chapter presents the test examples that where created to verify the accuracy of the results produced by EFFECT. These case studies embrace all the finite element types and all the analysis types implemented in EFFECT.

This chapter is organized in four parts, one for each type of analysis:

1. Static linear analysis;
2. Static nonlinear analysis;
3. Design sensitivities analysis;
4. Structural optimization.

9.2. Static Linear Analysis

Since all the elements of EFFECT can go through a linear analysis, several examples with different levels of complexity were prepared to verify the coherence and accuracy of the solutions produced by EFFECT. The proposed verification examples are:

(1) Pratt truss bridge example (Truss2D);
(2) Small truss crane example (Truss3D);
(3) Beam frame example (Beam2D);
(4) Offshore Jacket example (Beam3D);
(5) Cantilever Beam example (Shell3Node– CST only);
(6) Square plate example (Shell3Node– DKT only);
(7) Cantilever channel section example (Shell3Node);
(8) Wing plate example (Shell3Node);
(9) Square plate example (Shell4Node– QKT only);
(10) Cantilever channel section example (Shell4Node);
(11) Wing plate example (Shell4Node);

All the results produced with EFFECT are compared with the ones obtained with ANSYS, with a 5 digit precision, by calculating the relative error. This is obtained by subtracting the quotient of the values that are being compared, from the unity, so that an error of 0 means the values are coincident.

9.2.1 Pratt Truss Bridge Example (Truss2D)

This example consists of a 2D truss structure built with 38 Truss2D elements and 20 nodes. The structure has 7 vertical concentrated loads applied at the nodes and 2 nodes are under restraints. The Pratt truss bridge model is shown in the figure 9.1.

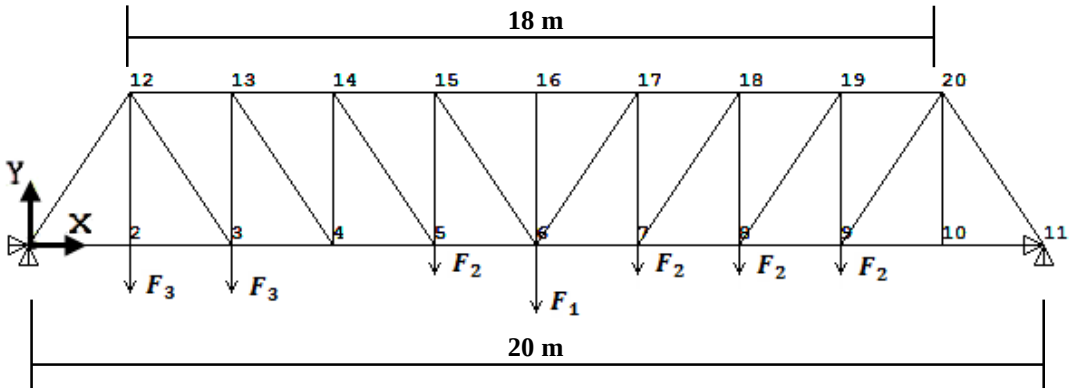

Figure 9.1 – Model of the Pratt truss bridge example, front view.

Geometric Data:

- Cross-section area, A **= 0.149** m^2**.**

Material Properties:

- Young´s Modulus**,** E **= 210 GPa.**

Load Values:

- 1 concentrated load of 3KN in the y-direction applied at node: 6. $F_1 = -3000$ **N;**
- 4 concentrated load of 1KN in the y-direction applied at nodes: 5, 7, 8 and 9. $F_2 = -1000$ **N;**
- 2 concentrated load of 2KN in the y-direction applied at nodes: 2 and 3. $F_3 = -2000$ **N**.

Boundary conditions:

- The nodes 1 and 11 are both **fixed**.

Table 9.1 presents the displacements results at 5 different nodes obtained with the LINK1 element of ANSYS and the Truss2D element of EFFECT, as well as the relative error in percentage. It is possible to see that the two elements produce the same results, for the 5 digit precision, shown in table 9.1.

Table 9.1 - Results for the Pratt truss bridge example, given by ANSYS and EFFECT.

NODE	ANSYS [LIINK1]		EFFECT [Truss2D]		ERROR(%)	
	UX (m)	UY (m)	UX (m)	UY (m)	UX	UY
2	-1.4914E-07	-2.9228E-06	-1.4914E-07	-2.9228E-06	0.00	0.00
5	-1.1931E-07	-9.1832E-06	-1.1931E-07	-9.1832E-06	0.00	0.00
6	1.2358E-07	-1.0009E-05	1.2358E-07	-1.0009E-05	0.00	0.00
8	4.8152E-07	-7.2988E-06	4.8152E-07	-7.2988E-06	0.00	0.00
14	1.4761E-06	-7.7202E-06	1.4761E-06	-7.7202E-06	0.00	0.00

9.2.2. Small Truss Crane Example (Truss3D)

In this example a crane built as a 3D truss structure with 390 Truss3D elements and 132 nodes is analyzed, subjected to 4 vertical concentrated loads applied at the nodes and restraints on 4 nodes. The truss crane model is shown in figure 9.2.

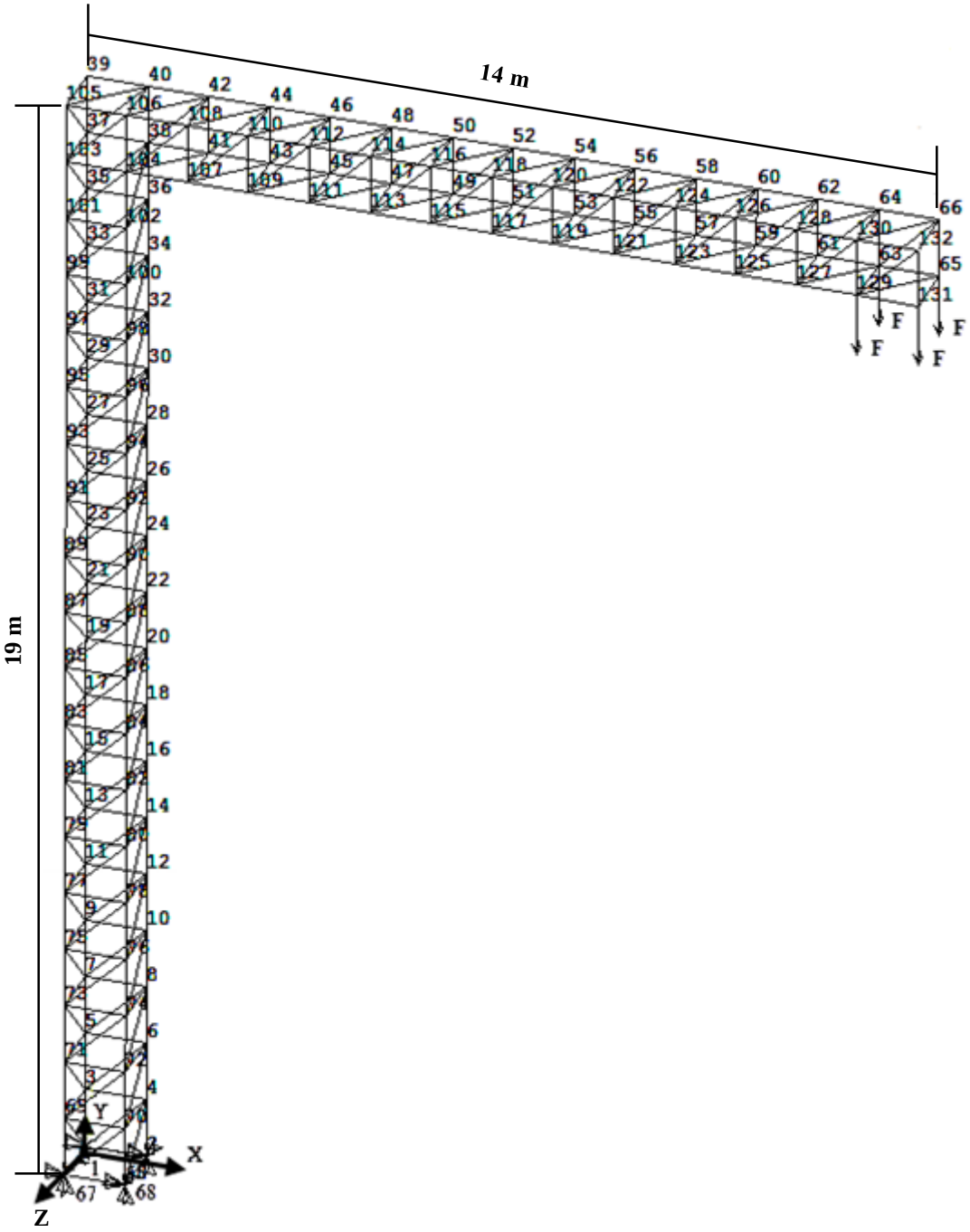

Figure 9.2 – Model of the small truss crane example, perspective view.

Geometric Data:

- Cross-section area, $A = 0.149\ m^2$.

Material Properties:

- Young´s Modulus**,** $\boldsymbol{E}$ **= 210 GPa.**

Load Values:

- 4 vertical concentrated loads of 5 KN in the y-direction applied at nodes: 63, 65, 129, and 131, $\boldsymbol{F = -5000\ N}$.

Boundary conditions:

- The nodes 1, 2, 67 and 68 are **fixed**.

Table 9.2 presents the displacements results at 7 different nodes obtained with the LINK8 element of ANSYS and the Truss3D element of EFFECT, as well as the relative error. Similarly to what has occurred in the previous example, the two elements produce the same results, for the 5 digit precision shown.

Table 9.2 – Results for the truss crane example, given by ANSYS and EFFECT.

NODE	ANSYS [LIINK8]			EFFECT [Truss3D]			ERROR(%)		
	UX (m)	UY (m)	UZ (m)	UX (m)	UY (m)	UZ (m)	UX	UY	UZ
10	6.7114E-05	-1.7258E-05	-1.7258E-05	6.7114E-05	-1.7258E-05	-1.7258E-05	0.00	0.00	0.00
30	8.1655E-04	-6.0403E-05	6.0403E-05	8.1655E-04	-6.0403E-05	6.0403E-05	0.00	0.00	0.00
45	1.3389E-03	-6.2130E-04	-8.7728E-05	1.3389E-03	-6.2130E-04	-8.7728E-05	0.00	0.00	0.00
65	1.3260E-03	-2.7306E-03	-1.0067E-04	1.3260E-03	-2.7306E-03	-1.0067E-04	0.00	0.00	0.00
87	4.1707E-04	3.9949E-05	-3.9949E-05	4.1707E-04	3.9949E-05	-3.9949E-05	0.00	0.00	0.00
116	1.5336E-04	-1.0098E-03	-5.2133E-05	1.5336E-04	-1.0098E-03	-5.2133E-05	0.00	0.00	0.00
132	1.5439E-03	-2.7174E-03	-3.3917E-05	1.5439E-03	-2.7174E-03	-3.3917E-05	0.00	0.00	0.00

9.2.3. Beam Frame Example (Beam2D)

This example represents a 4-bay 3-story frame with 74 nodes and 81 Beam2D elements. Distributed loads are applied on 36 elements and 4 nodes are restrained, as shown in figure 9.3.

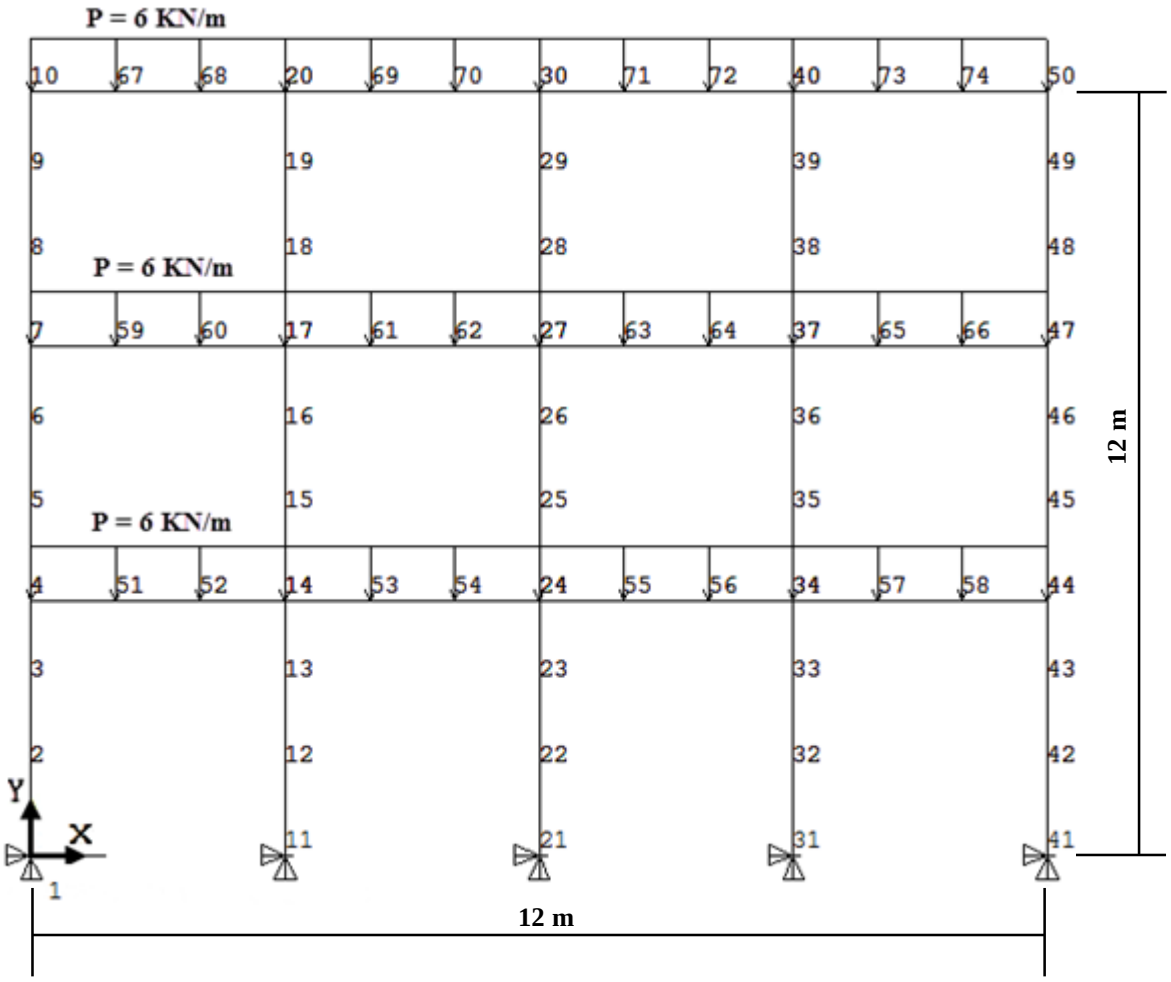

Figure 9.3 – Model of the beam frame example, front view.

Geometric Data:

- Cross-section area, $A = 0.1136\ m^2$;
- Inertia about the z-axis, $I_Z = 0.06056\ m^2$.

Material Properties:

- Young´s Modulus, $E = 210$ GPa.

Load Values:

- Distributed loads of ***6 KN/m***, applied on the horizontal elements between the nodes: 4 and 44; 7 and 47; 10 and 50.

Boundary conditions:

- The nodes 1, 11, 21, 31 and 41 are **fixed.**

The displacement results of 7 different nodes given by the BEAM3 element of ANSYS and the Beam2D element of EFFECT, as well as, the comparison between them, are presented in the table 9.3.

Table 9.3 – Results for the truss crane example, given by ANSYS and EFFECT.

NODE	ANSYS [BEAM3]			EFFECT [Beam2D]			ERROR(%)		
	UX (m)	UY (m)	ROTZ (rad)	UX (m)	UY (m)	ROTZ (rad)	UX	UY	RZ
4	-1.4914E-07	-2.9228E-06	-1.8284E-08	-1.4914E-07	-2.9228E-06	-1.8284E-08	0.00	0.00	0.00
16	-1.1931E-07	-9.1832E-06	-3.9524E-09	-1.1931E-07	-9.1832E-06	-3.9524E-09	0.00	0.00	0.00
19	1.2358E-07	-1.0009E-05	-2.2348E-08	1.2358E-07	-1.0009E-05	-2.2348E-08	0.00	0.00	0.00
34	4.8152E-07	-7.2988E-06	1.2433E-08	4.8152E-07	-7.2988E-06	1.2433E-08	0.00	0.00	0.00
50	1.4761E-06	-7.7202E-06	7.9574E-08	1.4761E-06	-7.7202E-06	7.9574E-08	0.00	0.00	0.00
69	-2.2312E-06	-2.2531E-06	-4.2425E-08	-2.2312E-06	-2.2531E-06	-4.2425E-08	0.00	0.00	0.00
72	-2.2312E-06	-2.2531E-06	4.2425E-08	-2.2312E-06	-2.2531E-06	4.2425E-08	0.00	0.00	0.00

9.2.4. Offshore Jacket Example (Beam3D)

This example represents a simplified model of an offshore structure, modeled with 74 nodes and 81 Beam3D elements. It has 15 horizontal concentrated loads applied and 6 nodes are under restraints. The beam frame model is shown in figure 9.4.

Geometric Data:

- Cross-section area, $\boldsymbol{A = 0.1136\ m^2}$**;**
- Inertia about the y-axis**,** $\boldsymbol{I_y = 0.06056\ m^2}$**;**
- Inertia about the z-axis**,** $\boldsymbol{I_Z = 0.06056\ m^2}$.

Material Properties:

- Young´s modulus, $\boldsymbol{E}$ **= 210 Gpa;**
- Poisson's ratio, $\boldsymbol{v}$ **= 0.3.**

Load Values:

- 15 horizontal concentrated loads of 4500 N in the y-direction applied on the nodes: 7, 10, 12, 15, 18, 20, 23, 26, 28, 31, 34, 36, 39, 42 and 44. $\boldsymbol{F = 4500\ N}$**;**

Boundary conditions:

- The nodes 1, 2, 3, 4, 5 and 6 are **fixed.**

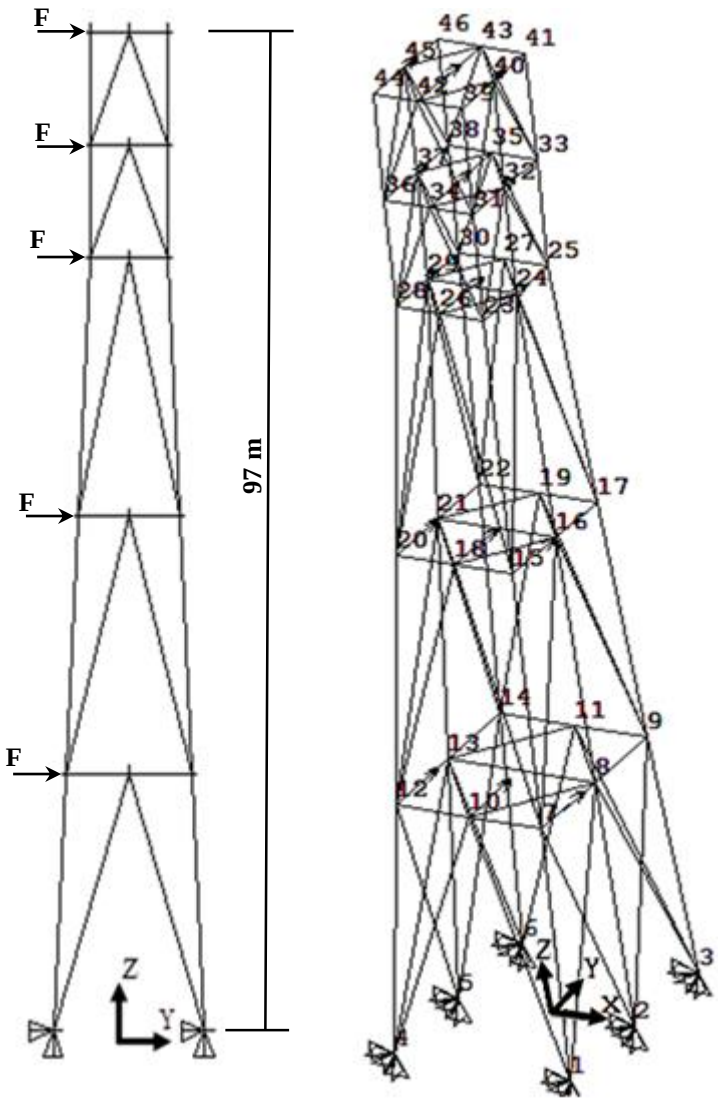

Figure 9.4 – Model of the offshore jacket example. Front view on the left and perspective view on the right.

The displacements results of 6 different nodes given by the BEAM4 element of ANSYS and the Beam3D element of EFFECT, as well as, the comparison between them, are presented in the tables 9.4 and 9.5.The table 9.4 presents the results of the translational degrees of freedom and the table 9.5presents the results of the rotational degrees of freedom.

Table 9.4 – Results of the translational degrees of freedom for the offshore jacket example, given by ANSYS and EFFECT.

Node	ANSYS [BEAM4]			EFFECT [Beam3D]			ERROR(%)		
	UX (m)	UY (m)	UZ (m)	UX (m)	UY (m)	UZ (m)	UX	UY	UZ
13	-3.5942E-07	9.7447E-05	-6.4412E-08	-3.5942E-07	9.7447E-05	-6.4412E-08	0.00	0.00	0.00
17	4.5844E-07	3.2247E-04	-6.9549E-05	4.5844E-07	3.2247E-04	-6.9549E-05	0.00	0.00	0.00
24	1.6104E-07	6.5351E-04	2.2987E-07	1.6104E-07	6.5351E-04	2.2987E-07	0.00	0.00	0.00
31	-4.3718E-08	7.9011E-04	6.3125E-05	-4.3718E-08	7.9011E-04	6.3125E-05	0.00	0.00	0.00
41	4.3323E-08	9.2173E-04	-6.3143E-05	4.3323E-08	9.2173E-04	-6.3143E-05	0.00	0.00	0.00
44	-1.4142E-08	9.2265E-04	6.3450E-05	-1.4142E-08	9.2265E-04	6.3450E-05	0.00	0.00	0.00

Table 9.5 - Results of the rotational degrees of freedom for the offshore jacket example, given by ANSYS and EFFECT.

Node	ANSYS [BEAM4]			EFFECT [Beam3D]			ERROR(%)		
	ROTX (rad)	ROTY (rad)	ROTZ (rad)	ROTX (rad)	ROTY (rad)	ROTZ (rad)	RX	UR	RZ
13	-4.1201E-06	1.3075E -12	1.9449E -07	-4.1201E-06	1.3075E -12	1.9449E-07	0.00	0.00	0.00
17	-1.1148E-05	2.5263E -06	5.8050E -08	-1.1148E-05	2.5263E -06	5.8050E-08	0.00	0.00	0.00
24	-1.1179E-05	-2.8738E-09	-1.2201E-07	-1.1179E-05	-2.8738E-09	-1.2201E-07	0.00	0.00	0.00
31	-1.1895E-05	-1.0139E-07	5.2752E -08	-1.1895E-05	-1.0139E-07	5.2752E-08	0.00	0.00	0.00
41	-1.1691E-05	-1.9639E-08	2.0111E -08	-1.1691E-05	-1.9639E-08	2.0111E -08	0.00	0.00	0.00
44	-1.1656E-05	-1.4503E-08	-1.9268E-08	-1.1656E-05	-1.4503E-08	-1.9268E-08	0.00	0.00	0.00

9.2.5. Cantilever Beam Example (Shell3Node – CST only)

This example was taken from the thesis of Kansara [17] and is meant for studying only the membrane behavior, due to CST element, of the Shell3Node element. A cantilever beam with a point load at the tip is modeled with 8 Shell3Node elements and 10 nodes. With such a crude discretization, the results are not comparable to the exact analytical solution, but match perfectly those presented by Kansara. The cantilever model is represented in figure 9.5.

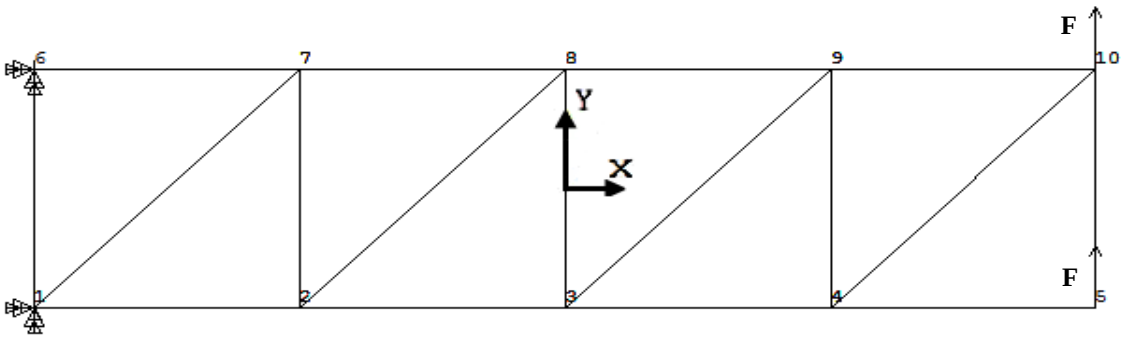

Figure 9.5 - Model of the cantilever beam example, front view.

Geometric Data:

- Length, ***L* = 48 in;**
- Height, ***h* = 24 in;**
- Thickness, ***t* = 2in.**

Material Properties:

- Young´s modulus, ***E* = 30000 ksi;**
- Poisson's ratio, ***v* = 0.25.**

Load Values:

- 2 vertical concentrated loads of 20 kpi in the y-direction applied on the nodes 5 and 10, $\boldsymbol{F = 20\ kips}$.

Boundary conditions:

- The nodes 1 and 6 are **fixed**.

The displacements results given by the Shell3Node element, for the membrane degrees of freedom, of EFFECT are compared, in the table 9.6, with the results presented by Kansara [15] for the same test example.

Table 9.6 - Results for the cantilever beam example, given by EFFECT and Kansara [15].

Node	**EFFECT [Shell3Node]**		**Kansara [15]**		**ERROR(%)**	
	UX (m)	UY (m)	UX (m)	UY (m)	UX	UY
3	-1.4159E-02	9.0347E-02	-1.4159E-02	9.0347E-02	0	0
13	-1.0825E-06	3.0403E-06	-1.825E-06	3.0403E-06	0	0

9.2.6. Square Plate Example (Shell3Node – DKT only)

This example was adpated, with some modifications, from Ferreira [50] representing a square plate and is used for studying only the plate bending action, due to DKT element, of the Shell3Node element. The square plate is modeled with 800 Shell3Node elements and 441 nodes. Vertical loads are applied on all nodes and the nodes on the four edges of the plate are simply supported. The square plate model is shown in the figures 9.6 and 9.7.

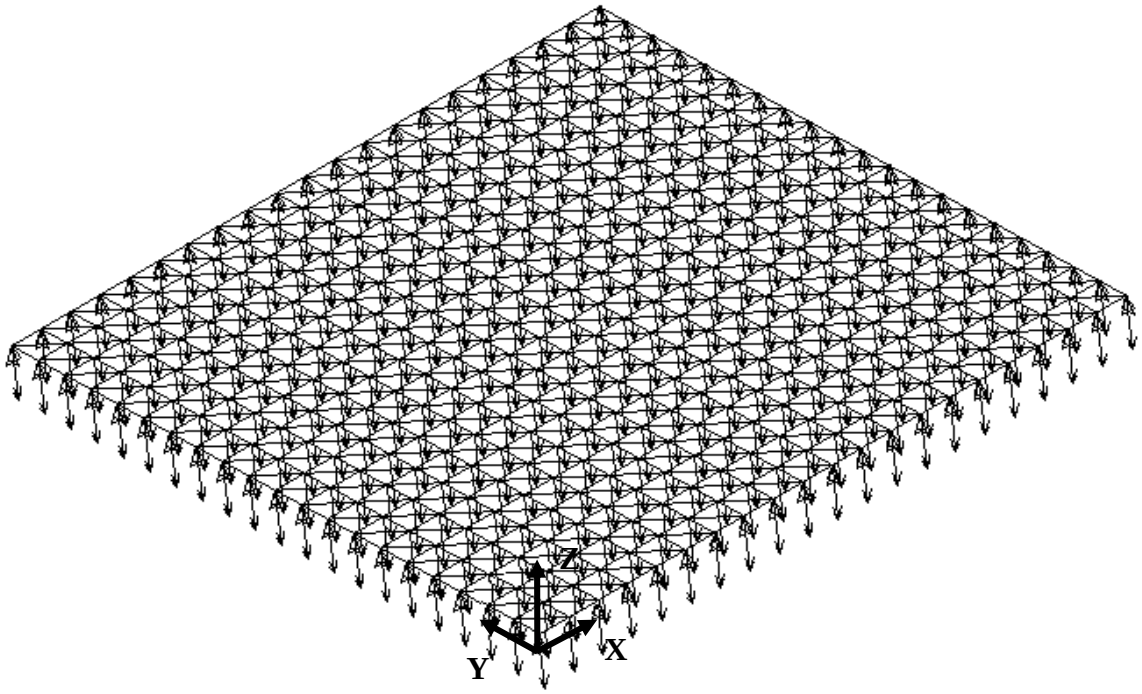

Figure 9.6 - Model of the square plate example, perspective view.

Figure 9.7 - Model of the square plate example, front view.

Geometric Data:

- Length, ***L* = 1m;**
- Height, ***h* = 1 m;**
- Thickness, ***t* = 0.01 m.**

Material Properties:

- Young´s modulus, ***E* = 210 Gpa;**
- Poisson's ratio, ***v* = 0.3.**

Load Values:

- Vertical concentrated loads of **-100 N** in the y-direction applied on all nodes.

Boundary conditions:

- The nodes from the 4 edges of the plate are **simply supported**.

The displacements of 6 different nodes obtained by the SHELL63 element of ANSYS and the Shell3Node element of EFFECT, as well as the relative error between them, are presented in the table 9.7.

Table 9.7 - Results for the square plate example, given by ANSYS and EFFECT.

Node	ANSYS [SHELL63]			EFFECT [Shell3Node]			ERROR(%)		
	UZ (m)	ROTX (rad)	ROTY (rad)	UZ (m)	ROTX (rad)	ROTY (rad)	UZ	RX	RY
136	-6.8500E-03	-1.4663E-02	3.0871E-03	-6.8500E-03	-1.4663E-02	3.0871E-03	0.00	0.00	0.00
178	-7.9643E-03	-7.5008E-03	3.6260E-03	-7.9643E-03	-7.5008E-03	3.6260E-03	0.00	0.00	0.00
201	-8.2468E-03	-3.7801E-03	-3.7801E-03	-8.2468E-03	-3.7802E-03	-3.7802E-03	0.00	0.00	0.00
264	-7.9643E-03	7.5008E-03	-3.6260E-03	-7.9643E-03	7.5008E-03	-3.6260E-03	0.00	0.00	0.00
327	-6.0322E-03	1.7973E-02	-2.6990E-03	-6.0322E-03	1.7973E-02	-2.6990E-03	0.00	0.00	0.00
348	-5.0556E-03	2.0995E-02	-2.2415E-03	-5.0556E-03	2.0995E-02	-2.2415E-03	0.00	0.00	0.00

9.2.7. Cantilever Channel Section Example (Shell3Node)

This example was adapted, with few modifications, from the thesis of Kansara [17] and is meant for studying the membrane and bending behavior of the Shell3Node element. A U-shape channel section modeled with 1200 Shell3Node elements and 631 nodes is subjected to 10 concentrated loads at one end while being fixed at the opposite extremity. The model is shown in figures 9.8 and 9.9.

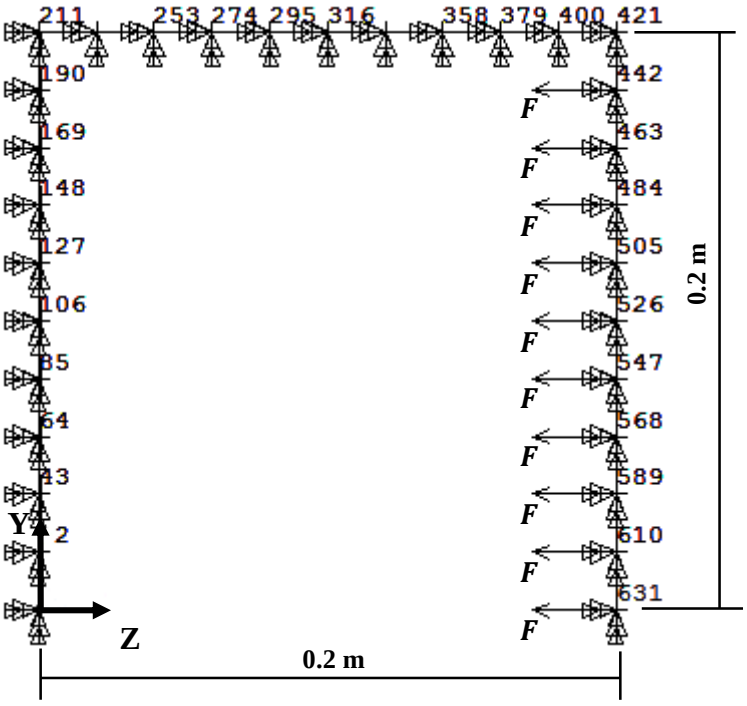

Figure 9.8 - Model of cantilever channel section example, left view.

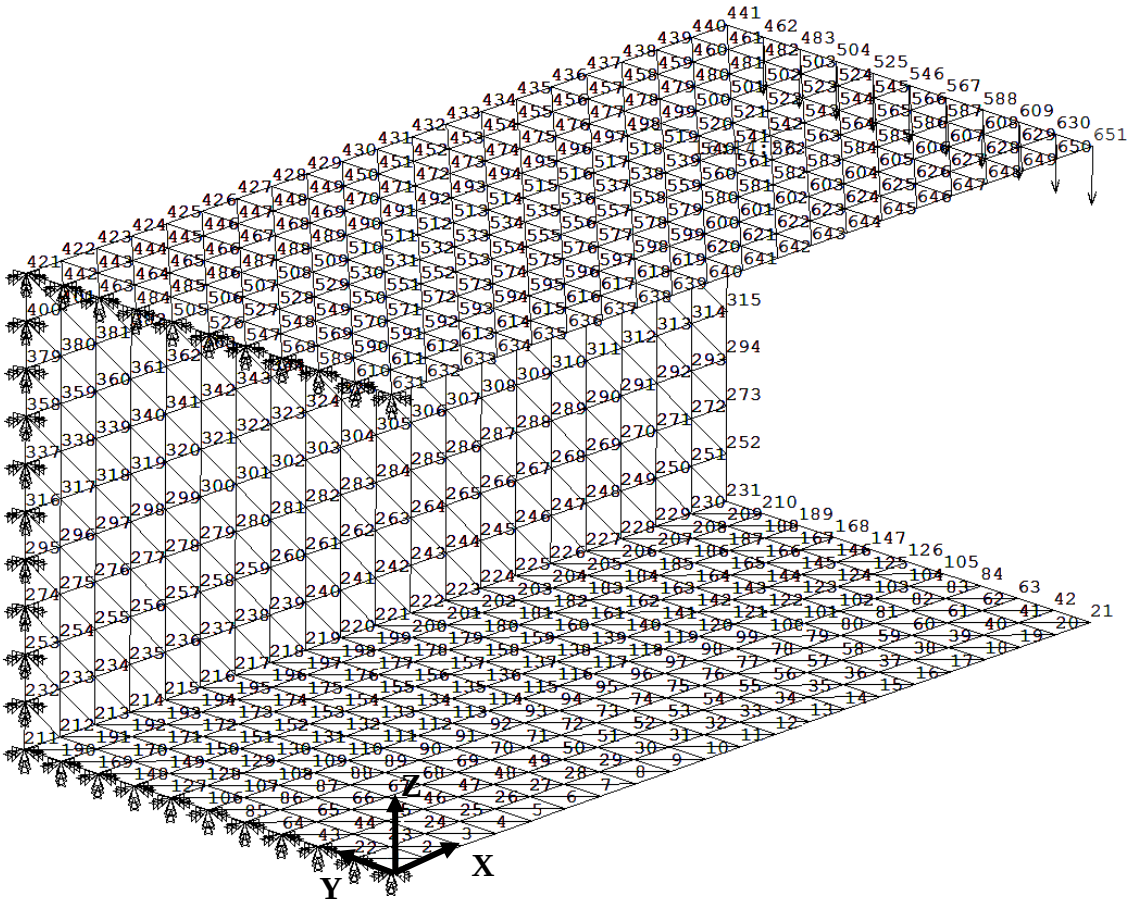

Figure 9.9 - Model of cantilever channel section example, perspective view.

Geometric Data:

- Thickness, t **= 0.001 m**
- Length, L **= 0.4 m**

Material Properties:

- Young´s modulus, E **= 210 GPa.**
- Poisson's ratio, ν **= 0.3.**

Load Values:

- 10 concentrated loads of 1 KN in the z-direction applied on the nodes 462, 483, 25, 504, 525, 546, 567, 588, 609, 630 and 651. $\boldsymbol{F = -1000}$ **N**;

Boundary conditions:

- The nodes from the left edge of the beam are **fixed**.

The displacements results of 6 different nodes given by the SHELL63 element of ANSYS and the Shell3Node element of EFFECT, as well as, the comparison between them, are presented in the tables 9.8 and 9.9.The table 9.8 presents the results of the translational degrees of freedom and the table 9.9 presents the results of the rotational degrees of freedom.

Table 9.8 - Results of the translational degrees of freedom for the cantilever channel section example, given by ANSYS and EFFECT.

Node	ANSYS [SHELL63]			EFFECT [Shell3Node]			ERROR(%)		
	UX (m)	UY (m)	UZ (m)	UX (m)	UY (m)	UZ (m)	UX	UY	UZ
546	3.8911E-05	-8.4139E-04	-2.9246	3.9042E-05	-8.3328E-04	-2.9224	0.34	0.97	0.07
567	-1.3687E-05	-8.3879E-04	-3.5525	-1.3052E-05	-8.3063E-04	-3.5501	4.87	0.98	0.07
588	-6.5823E-05	-8.3708E-04	-4.1570	-6.4707E-05	-8.2894E-04	-4.1545	1.73	0.98	0.06
609	-1.1814E-04	-8.3627E-04	-4.7337	-1.1649E-04	-8.2814E-04	-4.7311	1.42	0.98	0.06
630	-1.7093E-04	-8.3607E-04	-5.2834	-1.6877E-04	-8.2794E-04	-5.2807	1.28	0.98	0.05
651	-2.2431E-04	-8.3606E-04	-5.8145	-2.2163E-04	-8.2793E-04	-5.8117	1.21	0.98	0.05

Table 9.9 - Results of the rotational degrees of freedom for the cantilever channel section example, given by ANSYS and EFFECT.

NODE	ANSYS [SHELL63]		EFFECT [Shell3Node]		ERROR(%)	
	ROTX (rad)	ROTY (rad)	ROTX (rad)	ROTY (rad)	RX	RY
546	3.1805E+01	1.4616E+01	3.1797E+01	1.4599E+01	0.03	0.12
567	3.0833E+01	1.7183E+01	3.0826E+01	1.7167E+01	0.02	0.09
588	2.9508E+01	1.9357E+01	2.9502E+01	1.9341E+01	0.02	0.08
609	2.8086E+01	2.1108E+01	2.8080E+01	2.1093E+01	0.02	0.07
630	2.6877E+01	2.2437E+01	2.6871E+01	2.2422E+01	0.02	0.07
651	2.6451E+01	2.3463E+01	2.6444E+01	2.3448E+01	0.03	0.06

9.2.8. Wing Plate Example (Shell3Node)

This example was adpated, with some modifications, from Ferreira [50] and it is meant for studying the membrane and bending behavior of the Shell3Node element and consists of 154 Shell3Node elements and 96 nodes, representing a simplified model of an airplane wing. It has applied 16 concentrated loads and 6 nodes are under restraints. The wing plate model is shown in the figure 9.10 and 9.11.

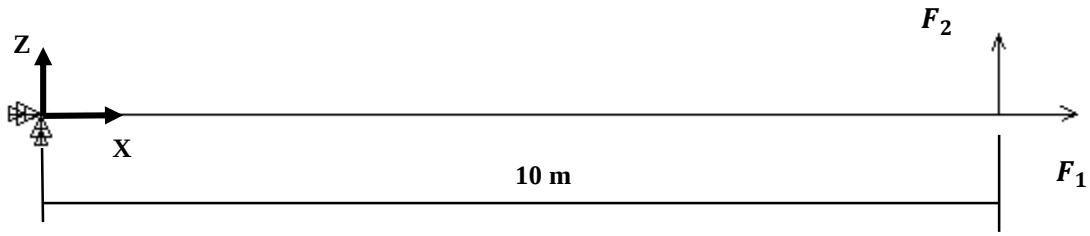

Figure 9.10 - Model of the wing plate example, top view.

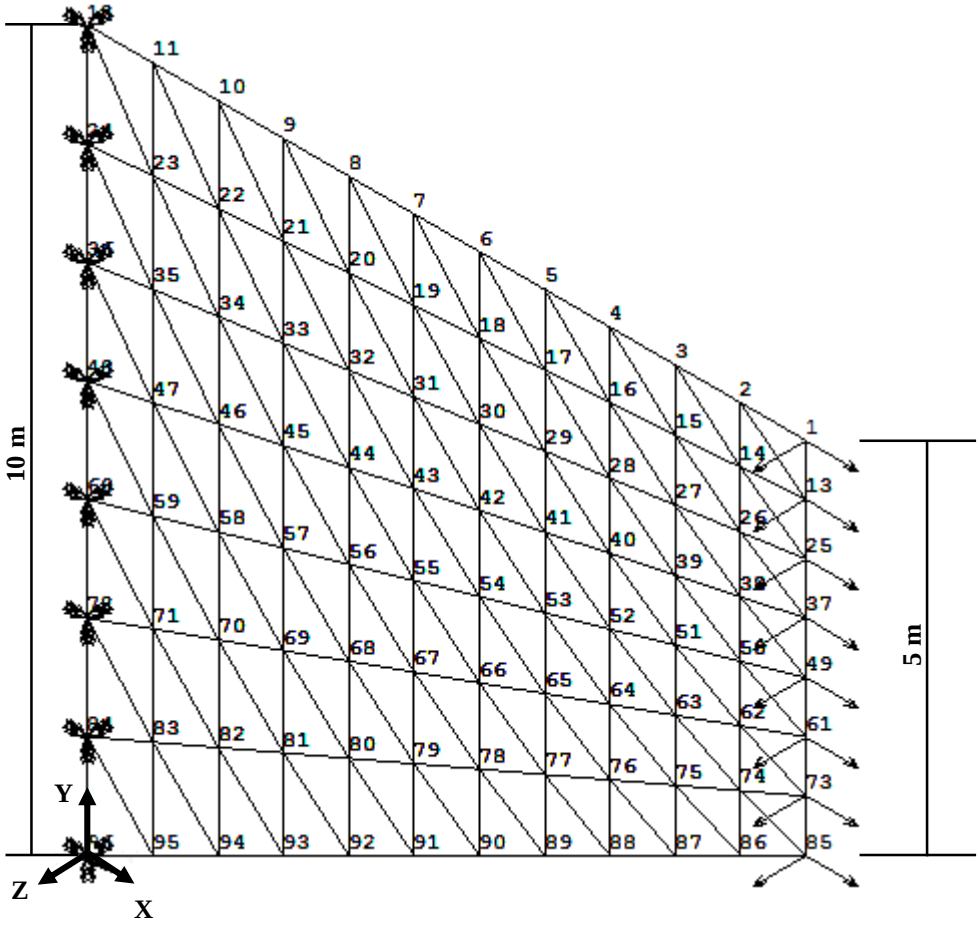

Figure 9.11 - Model of the wing plate example, perspective view.

Geometric Data:

- Thickness, ***t* = 1 m.**

Material Properties:

- Young´s modulus, **E = 210 Gpa;**
- Poisson's ratio, **v = 0.3.**

Load Values:

- 8 concentrated loads of 1 KN in the x-direction applied on the nodes 1, 13, 25, 37, 49, 61, 73 and 85. $\boldsymbol{F_1 = 1000}$ **N**;
- 8 concentrated loads of 1 KN in the z-direction applied on the nodes 1, 13, 25, 37, 49, 61, 73 and 85. $\boldsymbol{F_2 = 1000}$ **N**.

Boundary conditions:

- The nodes from the left edge of the wing: 12, 24, 36, 48, 60, 72 are **fixed**.

The displacements results of 6 different nodes given by the SHELL63 element of ANSYS and the Shell3Node element of EFFECT, as well as the relative error between them, are presented in tables 9.10 and 9.11. Table 9.10 presents the results of the translational degrees of freedom and table 9.11 presents the results of the rotational degrees of freedom.

Table 9.10 - Results of the translational degrees of freedom for the wing plate example, given by ANSYS and EFFECT.

Node	ANSYS [SHELL63]			EFFECT [Shell3Node]			ERROR(%)		
	UX (m)	UY (m)	UZ (m)	UX (m)	UY (m)	UZ (m)	UX	UY	UZ
13	8.5980E-08	-6.8071E-08	1.9255E-05	8.5981E-08	-6.8071E-08	1.9255E-05	0.00	0.00	0.00
25	7.7396E-08	-6.5650E-08	1.8987E-05	7.7396E-08	-6.5650E-08	1.8987E-05	0.00	0.00	0.00
37	7.0165E-08	-6.3671E-08	1.8688E-05	7.0166E-08	-6.3671E-08	1.8688E-05	0.00	0.00	0.00
49	6.3392E-08	-6.2156E-08	1.8354E-05	6.3392E-08	-6.2156E-08	1.8354E-05	0.00	0.00	0.00
61	5.6545E-08	-6.1085E-08	1.7984E-05	5.6545E-08	-6.1085E-08	1.7984E-05	0.00	0.00	0.00
73	4.9250E-08	-6.0337E-08	1.7581E-05	4.9250E-08	-6.0338E-08	1.7581E-05	0.00	0.00	0.00

Table 9.11 - Results of the translational degrees of freedom for the wing plate example, given by ANSYS and EFFECT.

NODE	ANSYS [SHELL63]		EFFECT [Shell3Node]		ERROR(%)	
	ROTX (rad)	ROTY (rad)	ROTX (rad)	ROTY (rad)	ROTX	ROTY
13	3.5856E-07	-2.9483E-06	3.5856E-07	-2.9483E-06	0.00	0.00
25	3.9551E-07	-2.9251E-06	3.9551E-07	-2.9251E-06	0.00	0.00
37	4.4226E-07	-2.8998E-06	4.4226E-07	-2.8998E-06	0.00	0.00
49	4.9254E-07	-2.8733E-06	4.9254E-07	-2.8733E-06	0.00	0.00
61	5.4159E-07	-2.8482E-06	5.4159E-07	-2.8482E-06	0.00	0.00
73	5.8503E-07	-2.8280E-06	5.8503E-07	-2.8280E-06	0.00	0.00

9.2.9. Cylindrical Shell Plate Example (Shell3Node)

This example was adapted from Takemoto and Cook [51] and consists of a hinged cylindrical shell subjected to a vertical concentrated load (P) at its center, as shown in figure 9.12. Due to symmetry, a model of ¼ of the plate was created with 800 Shell3Node elements and 441 nodes, as represented in figure 9.13, and symmetry boundary conditions were introduced.

Example data:

R = 2540 mm;
L = 254 mm;
Θ = 0.1 rad.

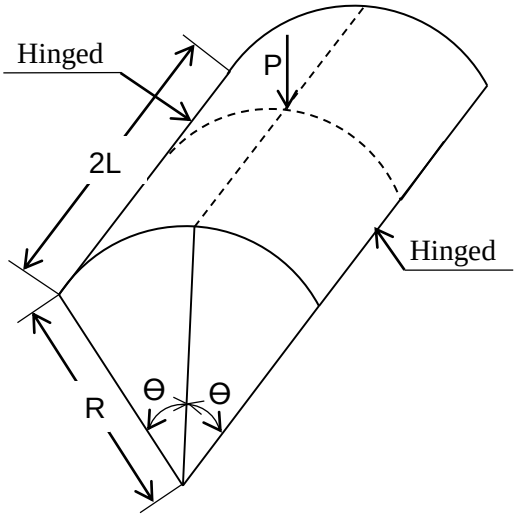

Figure 9.12- cylindrical shell model

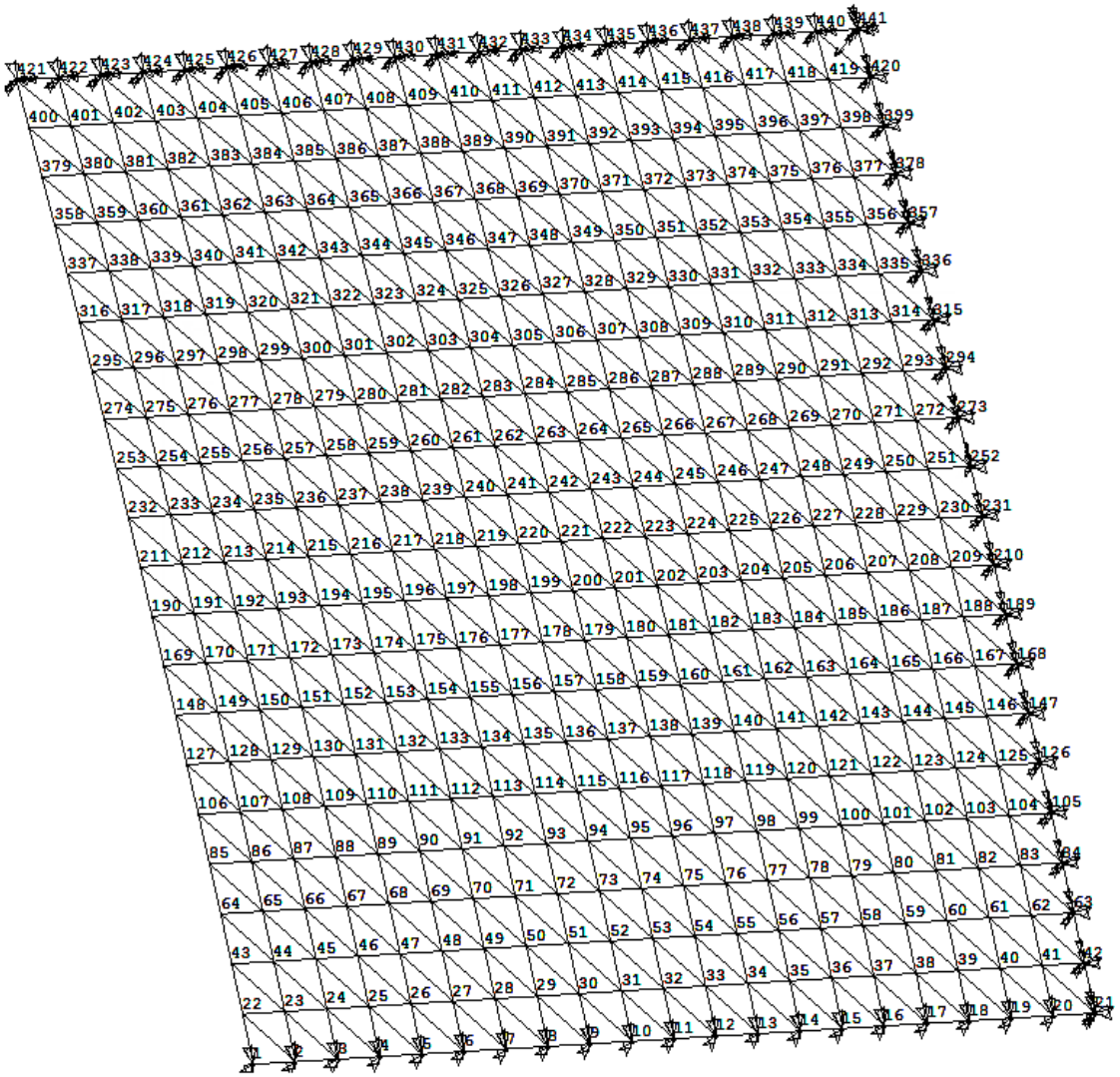

Figure 9.13 - Model of the wing plate example, perspective view.

Geometric Data:

- Thickness, ***t* = 6.35 mm;**
- Length, ***L* = 127 mm.**

Material Properties:

- Young´s modulus, **E = 3102.75 N/mm2**
- Poisson's ratio, **v = 0.3.**

Load Values:

- 1 concentrated load of 250N in the y-direction applied on the node 441. $\boldsymbol{P = -250}$ **N**.

Boundary conditions:

- Symmetry boundary conditions to the ones showed in the figure 9.12.

The displacements results of 6 different nodes, mostly from the edegs where the results are more relavant since the model studied is only ¼ of the plate presented in [], given by the SHELL63 element of ANSYS and the Shell3Node element of EFFECT, as well as the relative error between them, are presented in tables 9.12 and 9.13. Table 9.12 presents the results of the translational degrees of freedom and table 9.13 presents the results of the rotational degrees of freedom.

Table 9.12 - Results of the translational degrees of freedom for the cylindrical shell example, given by ANSYS and EFFECT.

Node	ANSYS [SHELL63]			EFFECT [Shell3Node]			ERROR(%)		
	UX (m)	UY (m)	UZ (m)	UX (m)	UY (m)	UZ (m)	UX	UY	UZ
127	4.2997E-02	5.7710E-01	7.5521E-02	4.2998E-02	5.7711E-01	7.5521E-02	0.00	0.00	0.00
184	5.7377E-03	-1.6693	3.3845E-02	5.7376E-03	-1.6692	3.3844E-02	0.00	0.00	0.00
221	3.9402E-04	-1.4972	4.8887E-02	3.9489E-04	-1.4971	4.8887E-02	0.22	0.00	0.00
231	-2.4627E-02	-3.2370	0.0000	-2.4626E-02	-3.2370	0.0000	0.00	0.00	0.00
431	0.0000	-4.1205	2.0692E-02	0.0000	-4.1205	2.0693E-02	0.00	0.00	0.00
441	0.0000	-1.0656E+01	0.0000	0.0000	-1.0656E+01	0.0000	0.00	0.00	0.00

Table 9.13 - Results of the translational degrees of freedom for the cylindrical shell example, given by ANSYS and EFFECT.

NODE	ANSYS [SHELL63]		EFFECT [Shell3Node]		ERROR(%)	
	ROTX (rad)	ROTZ (rad)	ROTX (rad)	ROTZ (rad)	ROTX	ROTZ
127	-9.5592E-03	-3.7566E-03	-9.5385E-03	-3.7565E-03	0.00	0.00
184	-1.0188E-02	3.4394E-02	-1.0170E-02	3.4394E-02	0.00	0.00
221	-1.8803E-02	2.5703E-02	-1.8778E-02	2.5704E-02	0.00	0.00
231	0.0000	5.3533E-02	0.0000	5.3534E-02	0.00	0.00
431	-4.8007E-02	0.0000	-4.8006E-02	0.0000	0.00	0.00
441	0.0000	0.0000	0.0000	0.0000	0.00	0.00

9.2.10. Square Plate Example (Shell4Node – QKT only)

This example is similar to the one presented in section 9.2.6 for Shell3Node. However in this case 400 Shell4Node elements and the same 441 nodes are used to model the square plate. All nodes are subjected to vertical concentrated loads and the 4 edges are simply supported. This example is meant for studying only the plate bending behavior, due to QKT element, of the Shell4Node element. The square plate model is shown in the figure 9.14.

Geometric Data:

- Length, ***L* = 1m;**
- Height, ***h* = 1 m;**
- Thickness, ***t* = 0.01 m.**

Material Properties:

- Young´s modulus, ***E* = 210 Gpa;**
- Poisson's ratio, ***v* = 0.3.**

Load Values:

- Vertical concentrated loads of **-100 N** in the y-direction applied on all nodes.

Boundary conditions:

- The nodes from the four edges of the plate are **simply supported**.

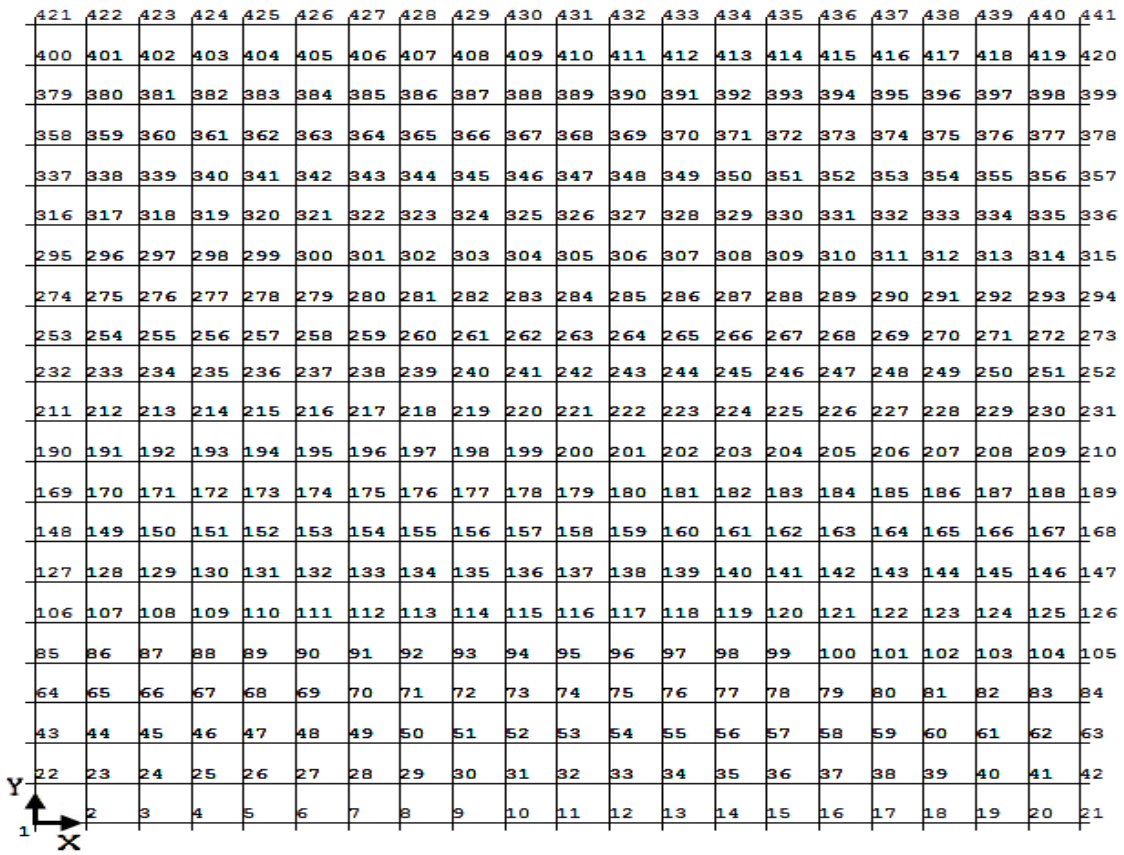

Figure 9.14 - Model of square plate example, front view.

The displacements results of 6 different nodes, given by the SHELL181 element of ANSYS and the Shell4Node element of EFFECT, as well as, the comparison between them, are presented in the table 9.14.

Table 9.14 - Results for the square plate example, given by ANSYS and EFFECT.

Node	ANSYS [SHELL181]			EFFECT [Shell4Node]			ERROR(%)		
	UZ	ROTX	ROTY	UZ	ROTX	ROTY	UZ	RX	RY
136	-6.8770E-03	-1.4753E-02	3.1284E-03	-6.9202E-03	-1.4771E-02	3.1622E-03	0.62	0.12	1.07
178	-7.9960E-03	-7.5490E-03	3.6589E-03	-8.0443E-03	-7.5559E-03	3.6890E-03	0.60	0.09	0.82
201	-8.2797E-03	-3.7944E-03	-3.7944E-03	-8.3292E-03	-3.7981E-03	-3.7981E-03	0.59	0.10	0.10
264	-7.9960E-03	7.5490E-03	-3.6589E-03	-8.0443E-03	7.5559E-03	-3.6890E-03	0.60	0.09	0.82
327	-6.0557E-03	1.8083E-02	-2.7430E-03	-6.0950E-03	1.8112E-02	-2.7324E-03	0.65	0.16	0.39
348	-5.0751E-03	2.1126E-02	-2.2876E-03	-5.1093E-03	2.1163E-02	-2.3290E-03	0.67	0.17	1.78

9.2.11. Cantilever Channel Section Example (Shell4Node)

This example is similar to the one presented in section 9.2.7 to verify the Shell3Node element. But now 600 Shell4Node and the same 631 nodes are used to model the cantilever channel section. The model of this example is shown in the figure 9.15 and 9.16.

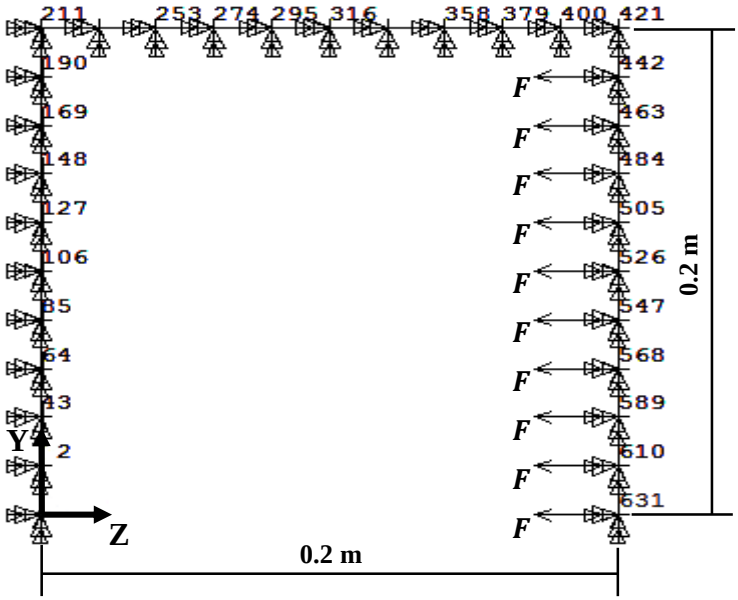

Figure 9.15 - Model of cantilever channel section example, left view.

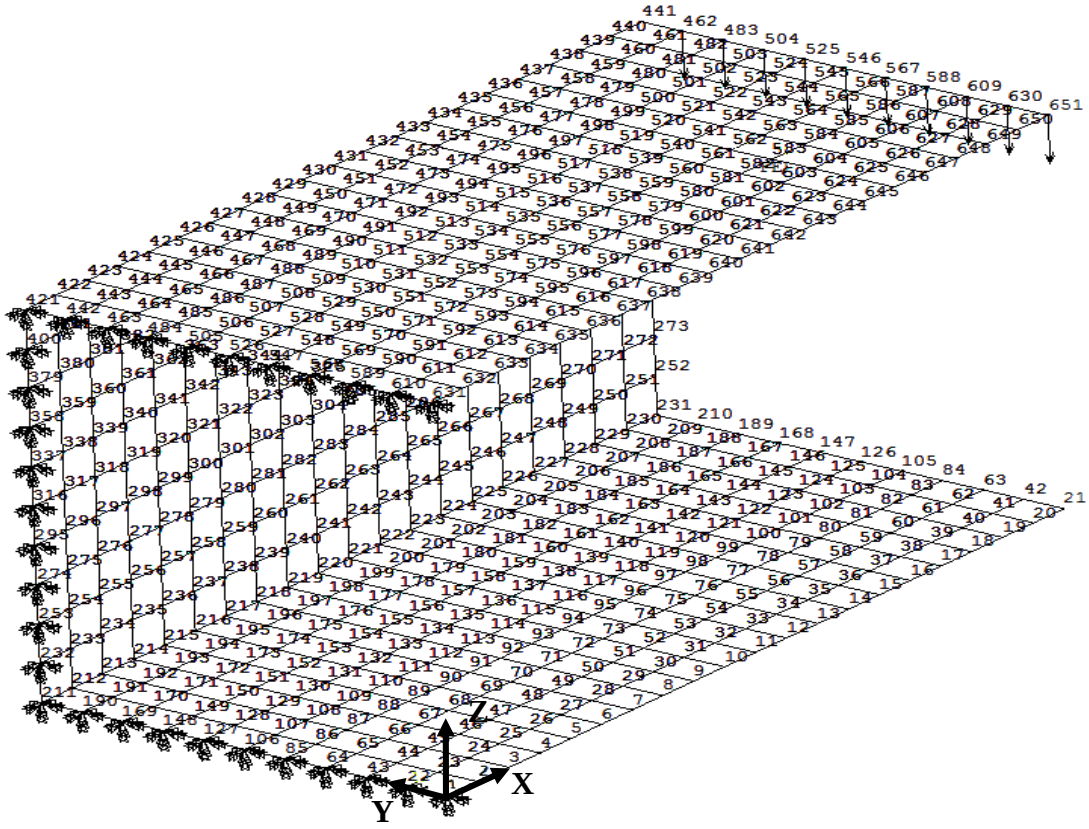

Figure 9.16 - Model of cantilever channel section example, perspective view.

Geometric Data:

- Thickness, t = **0.001 m**
- Length, L = **0.4 m**

Material Properties:

- Young´s modulus, E = **210 GPa.**
- Poisson's ratio, ν = **0.3.**

Load Values:

- 10 concentrated loads of 1 KN in the z-direction applied on the nodes 462, 483, 25, 504, 525, 546, 567, 588, 609, 630 and 651. $\boldsymbol{F = -1000}$ **N**;

Boundary conditions:

- The nodes from the left edge of the beam are **fixed**.

The displacements results of 6 different nodes given by the SHELL181 element of ANSYS and the Shell4Node element of EFFECT, as well as, the comparison between them, are presented in the tables 9.15 and 9.16.The table 9.15 presents the results for the translational degrees of freedom and the table 9.16 presents the results for the rotational degrees of freedom.

Table 9.15 - Results of the translational degrees of freedom for the cantilever channel section example, given by ANSYS and EFFECT.

Node	ANSYS [SHELL181]			EFFECT [Shell4Node]			ERROR(%)		
	UX (m)	UY (m)	UZ (m)	UX (m)	UY (m)	UZ (m)	UX	UY	UZ
546	4.9179E-05	-8.8475E-04	-2.9214	3.2023E-05	-9.3507E-04	-2.9200	53.57	5.38	0.05
567	-2.3704E-05	-8.7743E-04	-3.5498	-2.5996E-05	-9.3150E-04	-3.5487	8.82	5.80	0.03
588	-6.5514E-05	-8.7728E-04	-4.1548	-8.3679E-05	-9.2938E-04	-4.1541	21.71	5.61	0.02
609	-1.3051E-04	-8.7650E-04	-4.7317	-1.4170E-04	-9.2846E-04	-4.7315	7.90	5.60	0.00
630	-1.7736E-04	-8.7501E-04	-5.2809	-2.0046E-04	-9.2828E-04	-5.2816	11.52	5.74	0.01
651	-2.4135E-04	-8.7781E-04	-5.8111	-2.5984E-04	-9.2830E-04	-5.8132	7.12	5.44	0.04

Table 9.16 - Results of the rotational degrees of freedom for the cantilever channel section example, given by ANSYS and EFFECT.

NODE	ANSYS [SHELL181]		EFFECT [Shell4Node]		ERROR(%)	
	ROTX (rad)	ROTY (rad)	ROTX (rad)	ROTY (rad)	RX	RY
546	3.1899E+01	1.4736E+01	3.1855E+01	1.4701E+01	0.14	0.24
567	3.0930E+01	1.7268E+01	3.0876E+01	1.7272E+01	0.18	0.02
588	2.9561E+01	1.9465E+01	2.9543E+01	1.9461E+01	0.06	0.02
609	2.8121E+01	2.1210E+01	2.8105E+01	2.1215E+01	0.06	0.02
630	2.6804E+01	2.2505E+01	2.6837E+01	2.2525E+01	0.12	0.09
651	2.6208E+01	2.3318E+01	2.6271E+01	2.3414E+01	0.24	0.41

9.2.12. Wing Plate Example (Shell4Node Elements)

This example is similar as the one presented in section 9.2.8. Now 154 Shell4Node elements and 96 nodes are used to model the airplane wing. It has applied 16 concentrated loads and 6 nodes under restraints. The wing plate model is shown in the figure figure 9.17.

Geometric Data:

- Thickness, ***t* = 1 m**

Material Properties:

- Young´s modulus, **E = 210 GPa.**
- Poisson's ratio, **v = 0.3.**

Load Values:

- 8 concentrated loads of 1 KN in the x-direction applied on the nodes 1, 13, 25, 37, 49, 61, 73 and 85. $\boldsymbol{F_1 = 1000}$ **N**;
- 8 concentrated loads of 1 KN in the z-direction applied on the nodes 1, 13, 25, 37, 49, 61, 73 and 85. $\boldsymbol{F_2 = 1000}$ **N**;

Boundary conditions:

- The nodes from the left edge of the wing: 12, 24, 36, 48, 60, 72 are **fixed**.

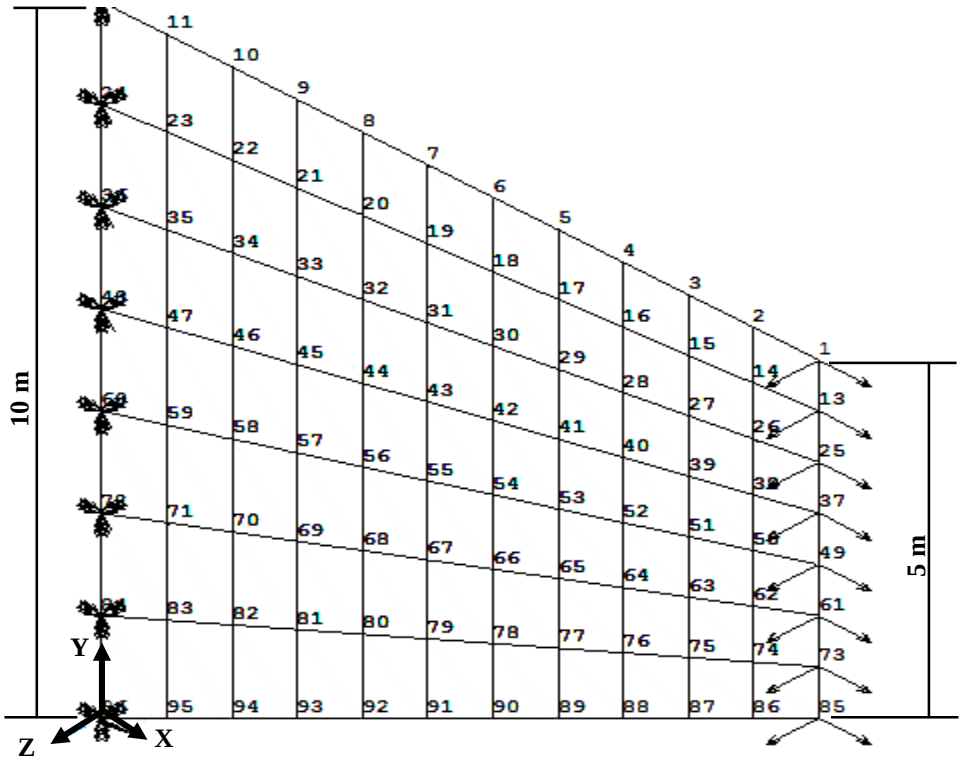

Figure 9.17 - Model of the wing plate example, perspective view.

The displacements results of 6 different nodes, located at the wing free edge, given by the SHELL181 element of ANSYS and the Shell4Node element of EFFECT, as well as the relative error between them, are presented in the tables 9.17 and 9.18. Table 9.17 presents the results of the translational degrees of freedom and table 9.18 the results of the rotational degrees of freedom.

Table 9.17 - Results of the translational degrees of freedom for the wing plate example, given by ANSYS and EFFECT.

Node	ANSYS [SHELL181]			EFFECT [Shell4Node]			ERROR(%)		
	UX (m)	UY (m)	UZ (m)	UX (m)	UY (m)	UZ (m)	UX	UY	UZ
13	8.1063E-08	-7.2700E-08	1.9582E-05	8.6345E-08	-7.0461E-08	1.9576E-05	6.12	3.18	0.03
25	8.1561E-08	-6.9445E-08	1.9297E-05	7.7777E-08	-6.7528E-08	1.9295E-05	4.86	2.84	0.01
37	6.9336E-08	-6.7711E-08	1.8985E-05	7.0409E-08	-6.5289E-08	1.8977E-05	1.52	3.71	0.04
49	6.4330E-08	-6.5610E-08	1.8641E-05	6.3516E-08	-6.3718E-08	1.8630E-05	1.28	2.97	0.06
61	5.7271E-08	-6.4962E-08	1.8264E-05	5.6506E-08	-6.2695E-08	1.8259E-05	1.35	3.62	0.03
73	4.6926E-08	-6.4743E-08	1.7857E-05	4.8872E-08	-6.2093E-08	1.7844E-05	3.98	4.27	0.07

Table 9.18 - Results of the rotational degrees of freedom for the wing plate example, given by ANSYS and EFFECT.

NODE	ANSYS [SHELL181]		EFFECT [Shell4Node]		ERROR	
	ROTX	ROTY	ROTX	ROTY	ROTX	ROTY
13	3.75E-07	-2.98E-06	3.80E-07	-2.99E-06	1.22	0.24
25	3.99E-07	-2.96E-06	3.98E-07	-2.96E-06	0.17	0.04
37	4.39E-07	-2.93E-06	4.36E-07	-2.93E-06	0.79	0.02
49	4.85E-07	-2.90E-06	4.79E-07	-2.89E-06	1.19	0.02
61	5.31E-07	-2.87E-06	5.23E-07	-2.86E-06	1.52	0.08
73	5.65E-07	-2.85E-06	5.56E-07	-2.85E-06	1.65	0.14

9.2.13. Cylindrical Shell Plate Example (Shel4Node Elements)

This example is similar as the one presented in section 9.2.9. Now 400 Shell4Node elements and 441 nodes are used to model ¼ of the cylindrical shell subjected to a vertical concentrated load at its center, as shown in the figure 9.18.

Geometric Data:

- Thickness, ***t* = 6.35 mm;**
- Length, ***L* = 127 mm.**

Material Properties:

- Young´s modulus, **E = 3102.75 N/mm2**
- Poisson's ratio, **v = 0.3.**

Load Values:

- 1 concentrated load of 250N in the y-direction applied on the node 441. $\boldsymbol{P = -250}$ **N**.

Boundary conditions:

- Symmetry boundary conditions are show in figure 9.12.

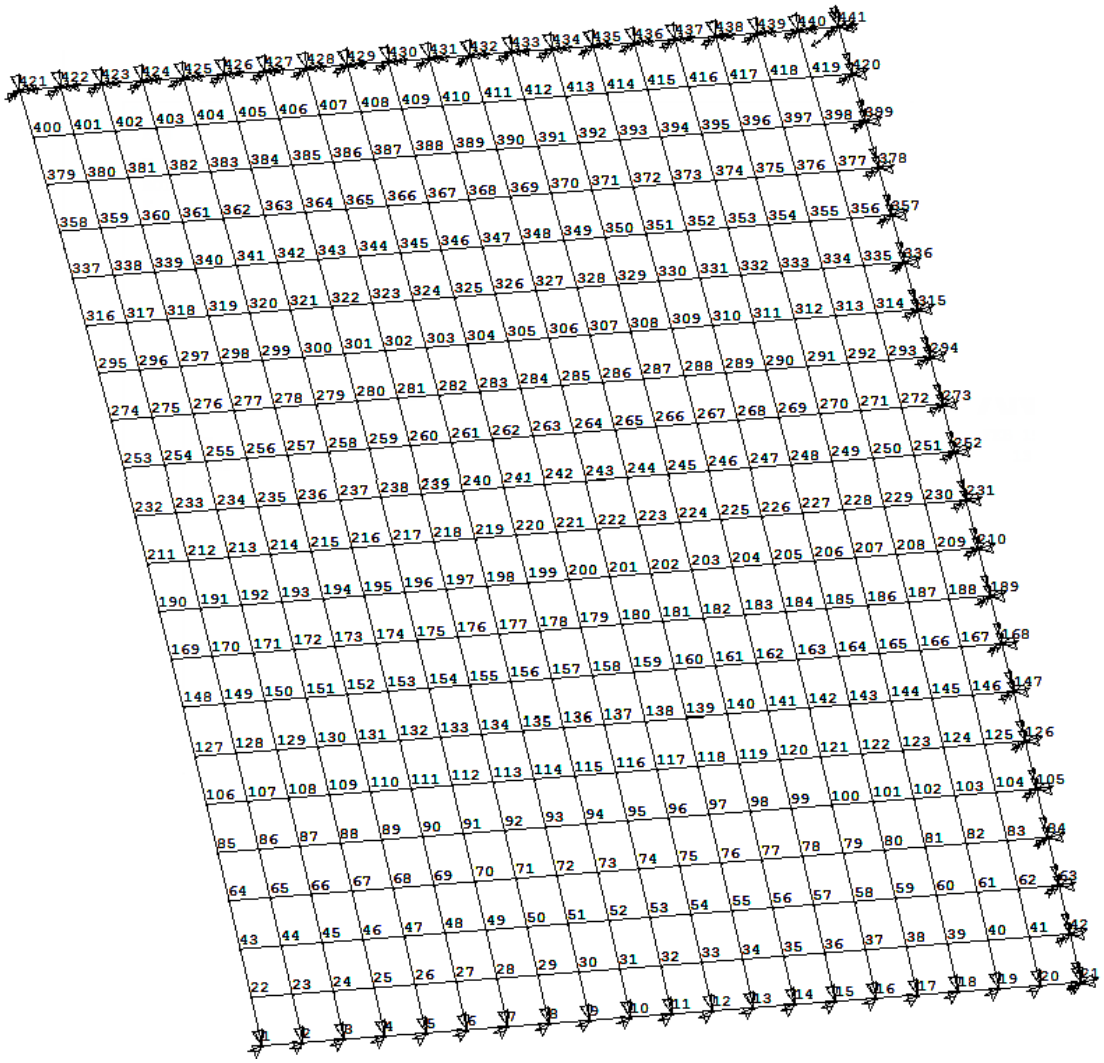

Figure 9.18 - Model of the wing plate example, perspective view.

The displacements results of 6 different nodes given by the SHELL181 element of ANSYS and the Shell4Node element of EFFECT, as well as the relative error between them, are presented in the tables 9.19 and 9.20. Table 9.19 presents the results of the translational degrees of freedom and table 9.20 presents the results of the rotational degrees of freedom.

Table 9.19 - Results of the translational degrees of freedom for the cylindrical shell example, given by ANSYS and EFFECT.

Node	ANSYS [SHELL181]			EFFECT [Shell4Node]			ERROR(%)		
	UX (m)	UY (m)	UZ (m)	UX (m)	UY (m)	UZ (m)	UX	UY	UZ
117	2.540E-02	-3.836E-01	6.101E-02	2.457E-02	-3.941E-01	6.094E-02	3.38	2.66	0.12
184	5.505E-03	-1.679	3.403E-02	6.550E-03	-1.660	3.400E-02	15.96	1.12	0.09
221	4.010E-04	-1.502	4.911E-02	-6.311E-04	-1.522	4.909E-02	163.54	1.29	0.06
231	-2.491E-02	-3.270	0.000	-2.616E-02	-3.270	0.000	4.78	0.02	0.00
431	0.000	-4.115	2.016E-02	0.000	-4.147	2.026E-02	0.00	0.78	0.49
441	0.000	-1.070E+01	0.000	0.000	-1.078E+01	0.000	0.00	0.74	0.00

Table 9.20 - Results of the rotational degrees of freedom for the cylindrical shell example, given by ANSYS and EFFECT.

NODE	ANSYS [SHELL181]		EFFECT [Shell4Node]		ERROR(%)	
	ROTX (rad)	ROTZ (rad)	ROTX (rad)	ROTZ (rad)	ROTX	ROTZ
117	-5,923E-03	1,271E-02	-5,978E-03	1,271E-02	0,91	0,02
184	-1,030E-02	3,441E-02	-1,032E-02	3,443E-02	0,15	0,08
221	-1,890E-02	2,570E-02	-1,905E-02	2,572E-02	0,81	0,10
231	0.0000	5,356E-02	0.0000	5,352E-02	0.00	0,06
431	-4,812E-02	0.0000	-4,814E-02	0.0000	0,04	0.00
441	0.0000	0.0000	0.0000	0.0000	0.00	0.00

9.3. Static Nonlinear Analysis

To verify the results of the nonlinear analysis a cantilever beam subjected to an applied moment at the free tip is analyzed. The example is considered in Urthaler and Reddy [52]. 20 Beam2D elements and 21 nodes were used to model the beam, as shown in the figure 9.19.

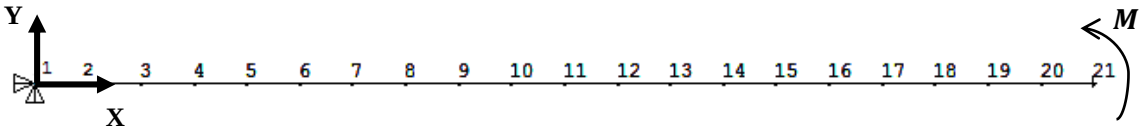

Figure 9.19 - Model of the cantilevered beam for a nonlinear analysis, front view.

Geometric Data:

- Length, $\boldsymbol{L = 100\ in}$**;**
- Cross-section area, $\boldsymbol{A = 0.25\ in^2}$**;**
- Inertia about the z-axis**,** $\boldsymbol{I_Z = 0,0052083\ in^4}$.

Material Properties:

- Young´s modulus, ***E*** **= 30E6 Psi;**

Load Values:

- A bending momentum about the z-axis applied on the node 21, ***M*****= 2454,354 lb×in.**

Boundary conditions:

- The node 1 is **fixed.**

The two elements produce the same results, for the 5 digit precision. In the table 9.21 are presented the displacements results, of the first 4 nodes counting from the free end of the beam, given by the BEAM3 element of ANSYS and the Beam2D element of EFFECT, as well as, the comparison between them

Table 9.21 - Results of the nonlinear analysis, given by ANSYS and EFFECT.

Node	ANSYS [BEAM3]			EFFECT [Beam2D]			ERROR(%)		
	UX	UY	ROTZ	UX	UY	ROTZ	UX	UY	RZ
18	-23.081	48.813	1.335	-23.081	48.813	1.335	0	0	0
19	-27.106	53.717	1.414	-27.106	53.717	1.414	0	0	0
20	-31.518	58.682	1.492	-31.518	58.682	1.492	0	0	0
21	-36.322	63.678	1.571	-36.322	63.678	1.571	0	0	0

9.4. Design Sensitivities Analysis

To verify the results of the sensitivity analysis a beam frame subjected to 1 horizontal concentrated load was analyzed. The model of the frame, shown in the figure 9.20 consists of 3 Beam2D elements and 4 nodes. The example is intended to calculate the sensitivities of 7 performances regarding 2 shape design variables.

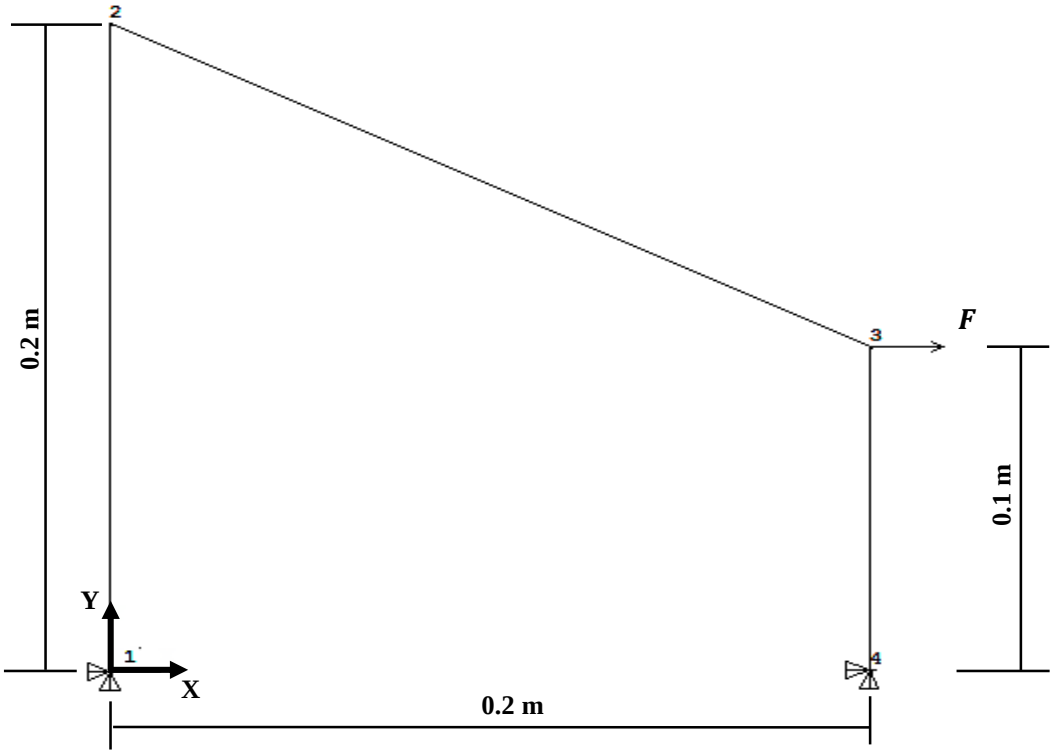

Figure 9.20 - Model of the beam frame for a sensitivity analysis, front view.

Geometric Data:

- Area, $\boldsymbol{A = 3.14159E-4\ m^2}$**;**
- Inertia about the z-axis**,** $\boldsymbol{I_Z = 7.85398E-9\ m^4}$.

Material Properties:

- Young´s modulus, ***E*** **= 210E9 Pa;**

Load Values:

- 1 concentrated loads of 200 N in the x-direction applied on the nodes 3, $\boldsymbol{F = 200\ N}$;

Boundary conditions:

- The node 1 is **fixed.**

Performances:

- Displacement of the node 2 in the x-direction: $\boldsymbol{\Psi 1 = 16.86239\ E-3\ m}$
- Displacement of the node 3 in the y-direction: $\boldsymbol{\Psi 2 = 16.86255\ E-3\ m}$
- Displacement of the node 2 in the x-direction: $\boldsymbol{\Psi 3 = 7.894115\ E-7\ m}$;
- Displacement of the node 3 in the y-direction: $\boldsymbol{\Psi 4 = -3.947058\ E-7\ m}$
- Stress at the midpoint section of the element 1: $\boldsymbol{\Psi 5 = 2.861908\ E+6\ Pa}$
- Stress at the midpoint section of the element 2: $\boldsymbol{\Psi 6 = 13.60763\ E+6\ Pa}$
- Stress at the midpoint section of the element 3: $\boldsymbol{\Psi 7 = -39.47058\ E+6\ Pa}$

Design Variables

- X-coordinate of node 2, $\boldsymbol{b_1 = 0.0\ m}$;
- Y-coordinate of node 2, $\boldsymbol{b_2 = 2.0\ m}$.

In tables 9.22 and 9.23 are presented the results of the sensitivity analysis. The values of the sensitivities obtained by the central finite differences method and the values of the sensitivities obtained by EFFECT, in the columns Ψ , ΔΨ(b), Ψ', respectively. The last column represents the difference between the values of ΔΨ(b) and Ψ'. The table 9.22 presents the values regarding the $\boldsymbol{b_1}$ design variable and the table 9.23 presents the values regarding the $\boldsymbol{b_2}$ design variable.

Table 9.22 – Sensitivities regarding the $\boldsymbol{b_1}$ design variable.

Perf.	Sensitivities regarding the $\boldsymbol{b_1}$ design variable		
	ΔΨ(b)	Ψ'	ERROR(%)
Ψ1	-8,43654E-03	-8,43719E-03	0,01
Ψ2	-4,22228E-03	-4,22223E-03	0,00
Ψ3	-8,43024E-03	-8,43079E-03	0,01
Ψ4	-2,02360E-07	-2,02364E-07	0,00
Ψ5	-6,88191E+06	-6,93816E+06	0,81
Ψ6	4,52698E+06	4,55187E+06	0,55
Ψ7	1,58672E+07	1,58676E+07	0,00

Table 9.23 – Sensitivities regarding the b_2 design variable.

Perf.	Sensitivities regarding the b_2 design variable		
	ΔΨ(b)	Ψ'	ERROR(%)
Ψ1	5,24492E-03	5,24435E-03	0,01
Ψ2	5,24294E-03	5,24296E-03	0,00
Ψ3	2,60578E-07	2,60585E-07	0,00
Ψ4	6,70655E-08	6,70602E-08	0,01
Ψ5	-6,19442E+06	-6,19443E+06	0,00
Ψ6	8,27529E+06	8,32520E+06	0,60
Ψ7	-1,52652E+07	-1,52654E+07	0,00

9.5. Structural Optimization

For testing the integration of EFFECT into a software system capable of performing structural optimization, the simple 31 Beam2D element structure shown in figure 9.24 was optimized. The same example was presented in Correia [49], where the details of this software system are described. The results obtained here are coincident with the ones presented in [49]. The example was also presented by Cardoso *et al.* [48]. In the initial configuration, the structure consists of 6 equal parts with 3 m length and 5 m high, with a total span of 18 m. The elements are grouped into 3 sets, and each one has independent cross-section. Table 9.25 shows the elements belonging to these groups.

The example considers the uncertainty of load values and the variability of material properties and cross-section dimensions through the use of random variables. As a consequence, structural safety is assessed using the FORM method of structural reliability analysis. The first step in order to establish the optimization problem consists in defining the stochastic model for the structure. In this case, 12 basic variables are used, including cross-section areas for the 3 sets, Y coordinates for the upper nodes, Young modulus and the load factor. This is associated with all applied loads shown in Fig. 9.24, where **P = 1MN**. The complete stochastic model is presented in table 9.26.

A set of performances is then defined. These are structural behavior measures computed by EFFECT used to define objective function, constraints or limit state functions (LSF). In this case total volume is considered a critical structural performance that should be optimized. Mid-span displacement must also be computed, as this value should be limited. As the 31 elements are grouped in just 3 sets, the stress in the most stressed elements for each set must also be

obtained. Table 9.27 shows all the performances defined for the 31 element truss, with the corresponding initial values.

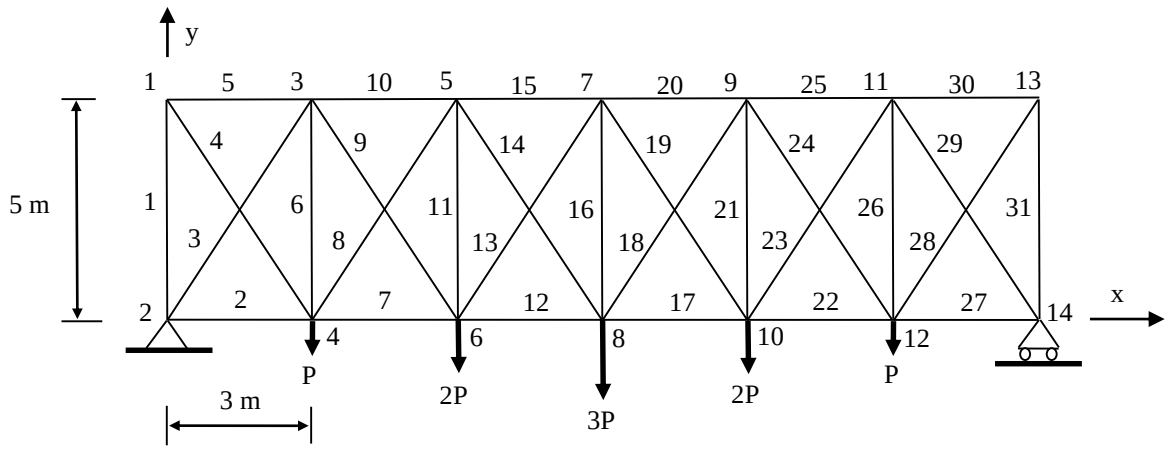

Table 9.24 - Model of the 31 element truss structure in the initial configuration.

Table 9.25 - Element sets for the truss structure.

Set	Elements
1	2, 7, 12, 17, 22, 27
2	3, 10, 15, 20, 25, 29
3	1, 4, 5, 6, 8, 9, 11, 13, 14, 16, 18, 19, 21, 23, 24, 26, 28, 30, 31

Table 9.26 – Stochastic model for the truss structure.

Var.	Description	Distribution	Average	Std. dev.
1	Cross-section area set 1	Normal	0.0314159 m^2	0.00314159 m^2
2	Cross-section area set 2	Normal	0.0314159 m^2	0.00314159 m^2
3	Cross-section area set 3	Normal	0.0314159 m^2	0.00314159 m^2
4	Node 1 Y coordinate	Normal	5 m	0.05 m
5	Node 3 Y coordinate	Normal	5 m	0.05 m
6	Node 5 Y coordinate	Normal	5 m	0.05 m
7	Node 7 Y coordinate	Normal	5 m	0.05 m
8	Node 9 Y coordinate	Normal	5 m	0.05 m
9	Node 11 Y coordinate	Normal	5 m	0.05 m
10	Node 13 Y coordinate	Normal	5 m	0.05 m
11	Young modulus	Normal	210 GPa	0.105 GPa
12	Load factor	Gumbel	1.0	0.25

Table 9.27 - Structural performances used in the RDO formulation.

Performance	Description	Initial value
1	Volume	+4.428744 m3
2	Vertical displacement at node 8	-1.730663×10^{-2} m
3	Stress at element 9	$+7.6247\times10^{7}$ Pa
4	Stress at element 12	$+1.6498\times10^{8}$ Pa
5	Stress at element 15	-1.7782×10^{8} Pa
6	Stress at element 17	$+1.6498\times10^{8}$ Pa
7	Stress at element 20	-1.7782×10^{8} Pa
8	Stress at element 23	$+7.6247\times10^{7}$ Pa

Seven limit state functions (LSF) are established, associated with the last 7 performances. Table 4 shows the corresponding reliability indexes, for the initial configuration in Fig. 9.24.First LSF is defined as $g(X) = \delta_{Adm} + \delta_{\text{Node}}$, 4th and 6th as $g(X) = \sigma_{Adm} + \sigma_{\text{Element}}$, respectively, as the underlying performances have negative values. The remaining are set as $g(X) = \sigma_{Adm} - \sigma_{\text{Element}}$. Allowable displacement is set to $\delta_{Adm} = L/500 = 36$ mm and allowable stress to $\sigma_{Adm} = 350$ MPa.

Table 9.28 - LSF and computed reliability indexes for the initial configuration.

LSF	Performance	Description	β
1	2	Vertical displacement at node 8	2.7899
2	3	Stress at element 9	5.1720
3	4	Stress at element 12	2.7594
4	5	Stress at element 15	2.5187
5	6	Stress at element 17	2.7594
6	7	Stress at element 20	2.5187
7	8	Stress at element 23	5.1720

Total volume is the objective function to be minimized with constrains on the reliability indexes, that should all be greater than $\beta_{\text{MIN}} = 3.7$. A total of 10 design variables are used. Three are associated with the mean values of the basic variables 1 to 3 in Table 2, related to the cross section areas of sets 1 to 3, respectively. The other are associated with the mean values of nodal Y coordinates for the top nodes 1, 3, 5, 7, 9, 11 and 13. All the design variables are presented in table 9.29, together with initial values and lower and upper bounds.

In order to solve the problem with the programs developed, input files describing the finite element analysis and the stochastic model are created. Also two MATLAB functions must be developed, to compute the objective function and the constraints. The *optimtool* optimization toolbox from MATLAB is used with the sequential quadratic programming (SQP) algorithm.

Table 9.29 - Design variables for the optimization problem formulations.

Design Variable	Description	Initial value	Lower bound	Upper bound
1	Cross-section area set 1	0.0314159 m^2	0.0113097 m^2	0.0804248 m^2
2	Cross-section area set 2	0.0314159 m^2	0.0113097 m^2	0.0804248 m^2
3	Cross-section area set 3	0.0314159 m^2	0.0113097 m^2	0.0804248 m^2
4	Node Y coord. 1	5 m	3.5 m	6.0 m
5	Node Y coord. 3	5 m	3.5 m	6.0 m
6	Node Y coord. 5	5 m	3.5 m	6.0 m
7	Node Y coord. 7	5 m	3.5 m	6.0 m
8	Node Y coord. 9	5 m	3.5 m	6.0 m
9	Node Y coord. 11	5 m	3.5 m	6.0 m
10	Node Y coord. 13	5 m	3.5 m	6.0 m

The Convergence for the optimization problem is obtained in 18 iterations, as shown in Fig. 9.21.Table 9.30 presents the initial and final values obtained for the design variables, objective function and constraints. Figure 9.22 represents the optimized configuration.

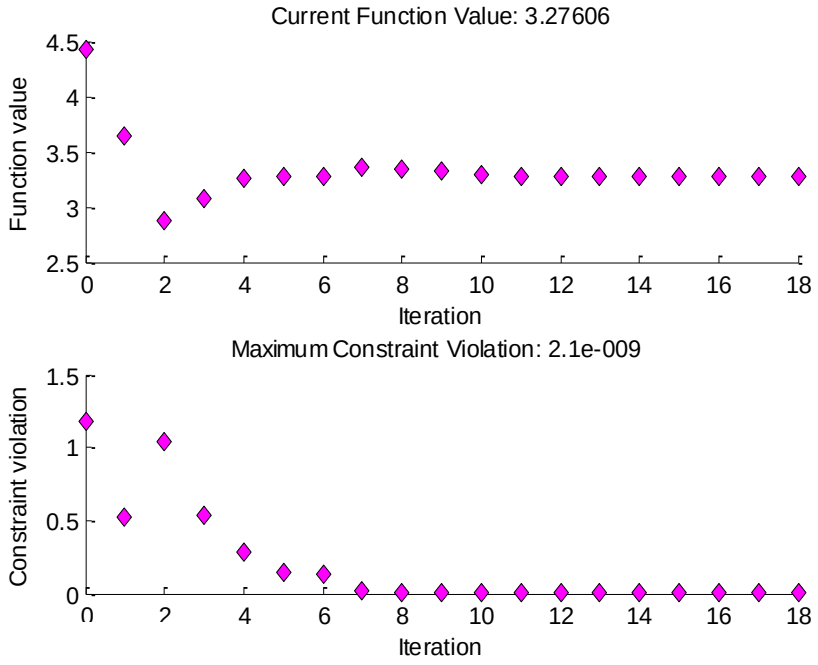

Figure 9.21 - Convergence history for the optimization problem.

Table 9.30 - Initial and optimized values of the optimization problem.

Description	Initial	Final
Design Var. 1 (m^2)	0.0314159	0.0393442
Design Var. 2 (m^2)	0.0314159	0.0514117
Design Var. 3 (m^2)	0.0314159	0.0148589
Design Var. 4 (m)	5.0	3.50000
Design Var. 5 (m)	5.0	3.63230
Design Var. 6 (m)	5.0	5.54038
Design Var. 7 (m)	5.0	6.00000
Design Var. 8 (m)	5.0	5.54038
Design Var. 9 (m)	5.0	3.63230
Design Var. 4 (m)	5.0	3.50000
Volume (m^3)	4.42874	3.27606
β_1	2.7899	3.7000
β_2	5.1720	3.7000
β_3	2.7594	4.1761
β_4	2.5187	4.6250
β_5	2.7594	4.1761
β_6	2.5187	4.6250
β_7	5.1720	3.7000

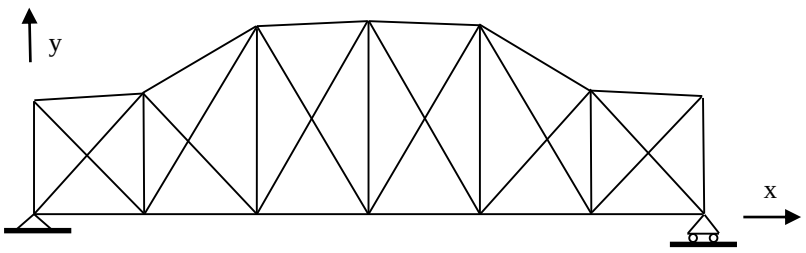

Figure 9.22 - Final configuration obtained .

Chapter 10

10. Conclusions and Future Developments

The main objective of this dissertation is the development of a program written in C++ language, EFFECT, to be used as a tool for performing finite element analysis, with the basic set of elements and capabilities implemented, and also to work as a solid foundation over which future developments could be easily derived. With this goal in mind, an assuming this is a first stage of development, a great deal of effort has been directed towards the creation of a robust framework and ensuring that all the functionalities and analysis (linear, nonlinear and sensitivities) implemented were prepared to be easily expanded. As explained in this thesis, not all the elements already present in EFFECT have their full range of functionalities operational and these functionalities and many other finite element types can be added in the future.

EFFECT architecture was designed to take profit from the features of C++ such as derived classes and polymorphism. To add a new element type only requires developing the corresponding class, and no restructuration or change to the remaining code is needed. In this stage 6 different element types were implemented: Truss2D, Truss3D, Beam2D, Beam3D, Shell3Node and Shell4Node elements. However only the static linear analysis is available for all of them, the sensitivities analysis is operational with the Truss2D and Beam2D elements and the static nonlinear analysis can be performed only with the Beam2D element. So the second stage of development should be the expansion of all the functionalities to all the elements already available in EEFECT. To test the accuracy of EFFECT's capabilities, a series of test examples were created and the results were mostly compared with the ones obtained with the commercial software ANSYS.

Regarding the static linear analysis results, obtained with Truss2D, Truss3D, Beam2D and Beam3D elements of EFFECT, a perfect match with ANSYS was found. The solutions calculated with Truss2D, Truss3D, Beam2D and Beam3D finite elements were compared with the ones computed with LINK1, LINK8, BEAM3 and BEAM4, respectively, from ANSYS. The Shell3Node element from EFFECT was compared against the 3 node version of ANSYS SHELL63, with the extra shape functions option deactivated and results were also very consistent, always presenting differences less than 0.1%, for the plate bending portion and less than 1% for the membrane portion. Regarding the Shell4Node element, results were very good for the plate bending part, with differences less than 1% for most of the cases when comparing with the SHELL181 element from ANSYS. However, the membrane portion did not perform well in any of the examples tested, with errors in the most part in the 3 to 6 % range and in some occurrences even exceeding 50% and 100%.

The Shell3Node and Shell4Node need to use a fictitious stiffness value associated with the in-plane rotation, the so-called drill degree of freedom, in order to prevent singularities in the stiffness matrix of the elements. This extra stiffness was considered in EFFECT but results for this degree of freedom are not present in the examples because it was not clear how ANSYS was computing it, and consequently, substantial differences were observed in the results. In future developments, the solution presented by Zienkiewicz and Taylor [11] for computing this fictitious stiffness may be implemented.

In conclusion, all the elements developed in this thesis were found to be accurate, in a static linear analysis, the biggest difference between EFFECT and ANSYS was observed in the Shell4Node element. It can also be concluded that the procedures involved in the assembly and the solution of the equations, work properly for all types of elements.

One of the subjects studied in this dissertation is the solution of nonlinear problems, more specifically, geometric nonlinear problems. To solve these types of problems a combination of the Newton-Raphson iterative method with the corotational kinematic formulation based on the work of Ranking and Brogan [25] was implemented. The corotational formulation was chosen because it allows, through the decomposing of the displacement field into a rigid body motion and a strain-producing displacement, to use the solvers already implemented for the linear analysis. In fact, due to the displacements decomposition, the small-strain theory to obtain the strains and element internal forces can still be used. Another reason for the implementation of the corotational formulation is the fact that it can be implemented independently of the finite elements, i.e., it is only necessary to implement the 2D and 3D version of the transformations needed to extract the strain-producing displacements from the total displacements for these 2 families of finite elements. The accuracy of the results obtained with nonlinear analysis done by EFFECT for the Beam2D element, which were compared with the ones obtained with ANSYS, is very good and allows concluding that all the nonlinear procedures are working correctly.

The third field of study pursuit in this thesis is the sensitivity analysis. It gives EFFECT the capability to calculate accurate values for gradients of performances with respect to design parameters through the continuum method of design sensitivity analysis and using the adjoint approach. This capability allows EFFECT to be integrated into a software system that can perform both structural optimization, reliability based design optimization (RDO) and robust design optimization (RDO) [48]. The results obtained with the sensitivity analysis were compared with the ones obtained with finite differences and it can be concluded that EFFECT is correctly calculating derivatives. The integration into the optimization software system requires the development of appropriate interfaces, which allow the communication between the optimization algorithms and the finite element code. In order to solve optimization problems, the commercial software MATLAB was used. The results obtained show that the interfaces developed are fully operational.

11. References

[1] Patzák, B., OOFEM project home page, 2011. <http://www.oofem.org>.

[2] Batoz, J. L., Bathe, K. J. and Ho, L. W., "A Study of Three-Noded Triangular Plate Bending Elements", International Journal for Numerical Methods in Engineering, Vol. 15, pp. 1771-1812, 1980.

[3] Batoz, J. L. and Tahar, M. B., "Evaluation of a New Quadrilateral Thin Plate Bending Element", International Journal for Numerical Methods in Engineering, Vol. 18, pp. 1655-1677, 1982.

[4] Turner, M. J., Clough, R. W., Martin, H. C. and Topp, L. J., "Stiffness and Deflection Analysis of Complex Structures", Journal of the Aeronautical Sciences, Vol. 23, pp.805-823, 1956.

[5] Clough, R. W., "The finite element method in plane stress analysis", Proc. 2nd A.S.C.E. Conf. in Electronic Computation, Pittsburgh, Pa., 1960.

[6] Courant, R., "Variational Methods for the Solution of Problems of Equilibrium and Vibration", Bulletin of American Mathematical Society, Vol. 49, pp. 1–23, 1943.

[7] Argyris, J. H., "Matrix Displacement Analysis of Anisotropic Shells by Triangular Elements", Journal of Royal Aero Society, Vol.69, pp. 801–805, 1965.

[8] Zienkiewicz, O. C., and Cheung, Y. K., "The Finite Element Method for Analysis of Elastic Isotropic and Anisotropic Slabs," Proceedings - Institution of Civil Engineers, Vol. 28, pp. 471–488, 1964.

[9] Cook, R. D., "Concepts and Applications of Finite Element Analysis", 2th edition, John Wiley & Sons, New York, NY, 2002.

[10] Bathe, K. J. and Ho, L. W., "A Simple and Effective Element for Analysis of General Shell Structures", Computers and Structures, Vol. 13, pp. 673-681, 1981.

[11] Zienkiewicz, O. C. and Taylor, R. L., "The Finite Element Method", 5th edition, Vol. 2, Butterworth-Heinemann, Oxford, 2000.

[12] Green, B. E., Strome, D. R. and Weikel, R. C., "Application of the stiffness method to the analysis of shell structures", Procedures on Aviation Conference, American Society of Mechanical Engineers, Los Angeles, 1961.

[13] Clough, R. W. and Tocher, J. L., "Finite Element Stiffness Matrices for Analysis of Plate Bending", Proc. Conference on Matrix Methods in Structural Mechanic, WPAFB, Ohio, pp. 515-545, 1965.

[14] Batoz, J. L., Bathe, K. J. and Ho, L. W., "A Study of Three-Noded Triangular Plate Bending Elements", International Journal for Numerical Methods in Engineering, Vol. 15, pp. 1771-1812, 1980.

[15] Irons, B. M., "Engineering Application of Numerical Integration in Stiffness Methods", The American Institute of Aeronautics and Astronautics, Vol. 4, No. 11, pp. 2035-2037, 1966.

[16] Ergatoudis, I., Irons, B. M. and Zeinkiewicz, O. C., "Curved Isoparametric 'Quadrilateral' elements for Finite Element Analysis", International Journal of Solids and Structures, Vol. 4, No.1, pp. 31-42, 1968.

[17] Kansara, K., "Development of Membrane, Plate and Flat Shell Elements in Java", Thesis submitted to the Faculty of the Virginia Polytechnic Institute & State University, 2004.

[18] McNeal, R. H., "A Simple Quadrilateral Shell Element", Computers and Structures, Vol. 8, pp. 175-183, 1978.

[19] Bathe, K. J. and Dvorkin, E. N., "A Four-Node Plate Bending Element Based on Mindlin/Reissner plate theory and a Mixed Interpolation", International Journal for Numerical Methods in Engineering, Vol. 21, pp. 367–383, 1985.

[20] Batoz, J. L. and Tahar, M. B., "Evaluation of a New Quadrilateral Thin Plate Bending Element", International Journal for Numerical Methods in Engineering, Vol. 18, pp. 1655-1677, 1982.

[21] Argyris, J. H., Balmer, H., Doltsinis, J. S., Dunne, P. C., Haase, M., Kleiber, M., Malejannakis, G. A., Mlejenek, H. P., MÄuller, M. and Scharpf, D. W., "Finite element method - the natural approach", Computer Methods in Applied Mechanics and Engineering, Vol.17-18, pp.1-106, 1979.

[22] Belytschko, T. and Glaum, L. W., "Applications of Higher Order Corotational Stretch Theories to Nonlinear Finite Element Analysis", Computers and Structures, Vol. 10, pp. 175–182, 1979.

[23] Wempner, G. A., "Finite elements, finite rotations and small strains of flexible shells", International Journal of Solids and Structures, Vol. 5, pp. 117–153, 1969.

[24] Belytschko, T. and Hsieh, B. J., "Nonlinear transient finite element analysis with convected coordinates", International Journal for Numerical Methods in Engineering, Vol. 7, pp. 255–271, 1973.

[25] Fraeijs de Veubeke, B. M., "The dynamics of flexible bodies", International Journal of Engineering Science, Vol. 14, pp. 895–913, 1976.

[26] Bergan, P. G. and Horrigmoe, G., “Incremental variational principles and finite element models for nonlinear problems”, Computer Methods in Applied Mechanical Engineering, Vol. 7, pp. 201-217, 1976.

[27] Rankin, C. C. and Brogan, F. A., “An element-independent Corotational procedure for the treatment of large rotations”, Pressure Vessel Technology, ASME, Vol. 108, pp. 165–174, 1986

[28] Felippa, C. A. and Haugan, B., “unified formulation of small-strain corotational finite elements: I. Theory”, Computer Methods in Applied Mechanics and Engineering, Vol. 194, pp. 2285–2335, 2005.

[29] Yves Dubois-Pèlerin, Pierre Pegon, “Object-oriented programming in nonlinear finite element analysis”, Structural Mechanics Unit, Joint Research Centre of the European Commission, Vol. 67, pp. 225-241, 1998

[30] Meyer, B., “Object-Oriented Software Construction”, 2nd edition, Prentice-Hall, Santa Barbara, CA, 1988.

[31] Cox, B. J., “Object-Oriented Programming: An Evolutionary Approach”, 2nd edition, Addison-Wesley, Boston, MA, 1986.

[32] Fenves, G. L. “Object-oriented programming for engineering software development”, Engineering With Computers, Vol. 6, pp. 1-15, 1990.

[33] Miller, G. R., “A LISP based, object-oriented approach to structural analysis”, Engineering with Computers, Vol. 4, pp. 197-203, 1988.

[34] Forde, B. W. R., Foschi, R. O., Stiemer, S. F., “Object-oriented finite element analysis”, Computers and Structures, Vol. 34, Issue 3, pp. 55-74, 1990.

[35] Zimmermann, T., Dubois-Pelerin, Y., Bomme, P., “Object-oriented finite element programming: I. Governing principles”, Computer Methods in Applied Mechanics and Engineering, Vol. 98, pp. 291–303, 1992.

[36] Miller, G. R., “An object-oriented approach to structural analysis and design” Computers and Structures, Vol. 40, pp. 75–82, 1991.

[37] Mackie, R. I., “Object oriented programming of the finite element method”, International journal of numerical methods in engineering, Vol. 35, pp. 425–436, 1992.

[38] Mackie, R. I., “Using objects to handle complexity in finite element software”, Engineering with Computers, Vol. 13, Issue 2, pp.99–111, 1997.

[39] Archer, G. C., Fenves, G., Thewalt, C., “A new object-oriented finite element analysis program architecture” Computers and Structures, Vol. 70, Issue. 1, pp. 63–75, 1999.

[40] Patzák, B. and Bittnar, Z., “Object oriented finite element modeling”, acta polytechnica, Vol. 39, No. 2, pp. 99–113, 1999.

[41] Dorn, W. S., Gomory, R. E. and Greenberg, H. J., "Automatic design of optimal structures", Journal de Mecanique, Vol. 3, 1964.

[42] Dobbs, M. W. and Felton, L. P., "Optmization of Truss Geometry", Journal of the Structural Division, ASCE, Vol.95, pp. 2105-2118, 1969.

[43] Vanderplaats, G. N. and Modes, F., "Automated design of trusses for optimum geometry", Journal of the Structural Division, ASCE, Vol.98, pp. 671-690, 1972.

[44] Haug, E. J., Choi, K. K., and Komkov, V., "Design sensitivity analysis of structural systems", Academic Press, New York, 1986.

[45] Twu, Sung-Ling and Choi, K. K., "Configuration design sensitivity analysis of built-up structures part I: Theory", International Journal for Numerical Methods in Engineering, Vol. 35, pp. 1127-1150, 1992.

[46] Cardoso, J. B., Phd Thesis, "Optimização de Configuração de Estruturas", Thesis submitted to Faculdade de Ciências e Tecnologia - UNL, 1994.

[47] Harrison, H. B., "Computer Methods in Structural Analysis", Prentice Hall Inc., New Jersey, 1973.

[48] Cardoso, J.B., Teixeira, A. P., Fraga, P.T., "Reliability-Based Design Optimization Using Design Sensitivity", 7th International Conference on Computational Stochastic Mechanics (CSM7), 15-18 June 2014, Santorini, Greece.

[49] Correia, M., Msc Thesis, "Optimização Robusta de Estruturas" (in Portuguese), FCT/UNL, 2014.

[50] Ferreira, A. J. M., "ELEMENTOS FINITOS em MATLAB", 1st edition, Fundação Calouste Gulbenkian, Lisboa, 1988.

[51] Takemoto, H. and Cook, R. D., "Some Modifications of an Isoparametric Shell Element", International Journal for Numerical Methods in Engineering, Vol. 7 No. 3, 1973.

[52] Reddy, J. N. and Urthale Y., "A corotational finite element formulation for the analysis of planar beams", COMMUNICATIONS IN NUMERICAL METHODS IN ENGINEERING, Vol. 21, pp. 553-570, 2005.