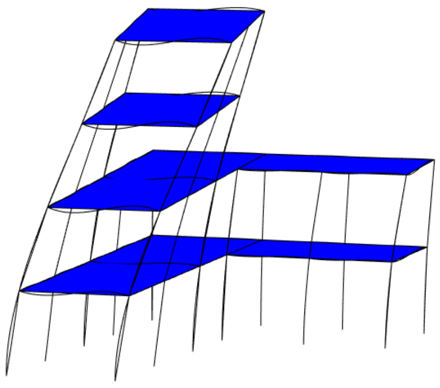


Static and Dynamic Analysis of Building Structures

Models with 3 Degrees of Freedom Per Story

João Salgado Canha Xavier Candeias

Thesis to obtain the Master of Science Degree in

Civil Engineering

Supervisor: Prof. Manuel da Cunha Ritto Corrêa

Examination Committee:

Chairperson: Prof. António Manuel Figueiredo Pinto da Costa

Supervisor: Prof. Manuel da Cunha Ritto Corrêa

Member of the Committee: Prof. Luís Manuel Coelho Guerreiro

May 2019

Declaration

ACKNOWLEDGEMENTS

The author would like to thank his supervisor, Prof. Manuel Corrêa, for his utmost patience and dedication, and for always being able to explain a new concept or idea in a simple manner.

He is deeply thankful to his parents for their unconditional trust and support throughout his academic journey, even in his moments of biggest doubt.

Finally, he would like to send a very special “thank you” to all his friends who, in one way or another, gave him the motivation and/or inspiration required to succeed.

RESUMO

A análise computacional de uma estrutura é muitas vezes um processo delicado e moroso, o que geralmente conduz a que esta seja efectuada apenas nas últimas etapas do processo de concepção de um edifício. Esta dissertação tem como objectivo a criação de um programa que conceda a uma equipa de projectistas a possibilidade de criar um modelo e fazer a sua análise estrutural desde as primeiras fases do projecto. O objectivo é portanto conceber um software que execute as análises estática e dinâmica de uma estrutura de uma forma expedita, utilizando algumas simplificações importantes, que permitam diminuir a complexidade do seu uso, sem perder de vista uma caracterização correcta dos aspectos mais importantes do comportamento estrutural de edifícios.

Com vista a abordar o problema de uma forma simples, considera-se que as estruturas a analisar têm apenas três graus de liberdade por piso. Três tipos de sub-estruturas são introduzidos, os quais podem ser combinados de modo a criar o modelo de um edifício. É dada a formulação da análise para cada tipo de sub-estrutura, detalhando-se também a análise dinâmica com recurso ao Eurocódigo 8.

É também apresentado um método para estimar os esforços e a quantidade de armadura necessária em cada elemento.

A implementação destes conceitos em MATLAB é apresentada, bem como a escolha da abordagem de programação por objectos.

Por fim, alguns testes são feitos, examinando-se os seus resultados de modo a fazer uma avaliação do programa desenvolvido.

Palavras-chave: Análise Estrutural de Edifícios, Interacção de Esforços, Modelação de Núcleos, Condensação Estática, Sub-Estruturas

ABSTRACT

Conventional structural analysis programs are quite often very detailed and time-consuming, and as such, their employment is reserved for late phases of a building project. The motivation for this dissertation comes from the desire to create a piece of software that enables structural analysis to be performed in early design stages, namely conceptual design. The goal is to write a code that can perform both static and dynamic analysis in an expeditious manner; capturing the most essential components of the behavior of structures, while not being overly complex, in order to extend the space of possibilities for any given project, which could help in the efficient design of buildings.

With the need for simplicity in the modelling as one of the main aspects of the work, structures are considered as having only three degrees of freedom per floor. Three different kinds of sub-structures are introduced, which can be combined to create the model of a building. The formulation for the analysis of each kind of sub-structure is given, as well as the detailed procedure for performing dynamic analysis of a structure.

A method for evaluating stresses is provided, which can then be used for estimating the reinforcement needed in all elements.

The implementation of these concepts into the MATLAB programming language is explained, as well as the object-oriented approach taken for the programming.

Finally, a few tests are performed, and its results are examined in order to check the validity of the software which was developed.

Keywords: Structural Analysis of Buildings, Stress Resultant Interaction, Core Wall Modelling, Static Condensation, Sub-Structures

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\boldsymbol{A}$	Transformation matrix from the local reference frame of an element to its substructure's frame.
$\boldsymbol{A}_{sg}^{j}$	Matrix relating the "Slave" degrees of freedom on floor j with the Global displacements.
$\boldsymbol{A}_{sg}$	Matrix relating the "Slave" degrees of a whole sub-structure with the Global displacements.
$\boldsymbol{A}_{w}$	Matrix relating the displacements on a wall (as a linear element) with the nodal displacements.
$\boldsymbol{C}$	Damping matrix of the model.
c_{cr}	Critical modal damping
$d^{\perp}$	Signed distance of a node to the origin of the global reference frame.
$\boldsymbol{f}_{g}^{(i)}$	Global forces vector of substructure i.
$\boldsymbol{f}_{s}^{*}$	Force vector of a sub-structure written only in terms of the "Slave" degrees of freedom.
$\boldsymbol{f}_{l}^{b}$	Force vector of a linear element in its local coordinate frame.
$\boldsymbol{f}_{e}^{b}$	Force vector of a linear element in its substructure's coordinate frame.
$\boldsymbol{f}_{f}$	Force vector of a whole 2D or 3D frame.
$\boldsymbol{f}_{p}$	Force vector of a wall.
$\boldsymbol{K}_{gg}^{(i)}$	Global stiffness matrix of substructure i.
$\boldsymbol{K}_{ss}^{*}$	Stiffness matrix of a sub-structure written only in terms of the "Slave" degrees of freedom.
$\boldsymbol{K}_{l}^{b}$	Stiffness matrix of a linear element in its local coordinate frame.
$\boldsymbol{K}_{e}^{b}$	Stiffness matrix of a linear element in its substructure's coordinate frame.
$\boldsymbol{K}_{f}$	Stiffness matrix of a whole 2D or 3D frame.
$\boldsymbol{K}_{p}$	Stiffness matrix of a wall.
$\boldsymbol{K}_{c}$	Stiffness matrix of a whole core wall.
$\boldsymbol{M}$	Mass matrix of the model.
$\boldsymbol{q}_{g}$	Global displacements vector.
$\boldsymbol{q}_{l}$	"Local" degrees of freedom vector of a sub-structure.
$\boldsymbol{q}_{s}$	"Slave" degrees of freedom vector of a sub-structure.
$\boldsymbol{q}_{l}^{b}$	Displacement vector of a linear element in its local coordinate frame.
$\boldsymbol{q}_{e}^{b}$	Displacement vector of a linear element in its substructure's coordinate frame.
$\boldsymbol{q}_{f}$	Displacement vector of a whole 2D or 3D frame.
$\boldsymbol{q}_{p}$	Displacement vector of a wall, seen as a linear element.
$\boldsymbol{q}_{G}$	Displacement vector in modal coordinates.
S_{d}	Spectral acceleration.
x_0, y_0	Values giving the position of a sub-structure's reference frame in terms of the global frame.
α_0	Angle between the reference frame of a sub-structure and the global reference frame.
β	Ratio used to define a "Connected" wall.
$\boldsymbol{\Lambda}$	Transformation matrix of beam displacements.
μ	Relative moment.
ν	Relative axial stress.
ω	Mechanical reinforcement ratio.

1 INTRODUCTION

1.1 Background and motivation

In the past few decades, big advances have been made regarding the development of tools for the design and structural analysis of buildings. The arrival of the digital era had such an immense influence that today almost any engineering project uses some kind of design software. However, most of these tools are oriented only at the more detailed and later stages of design, and they can require some time and learning to be used correctly. This results in the neglect of most structural aspects in the conceptual design phase, which is one of the first steps in a construction project, wherein the designing team must consider many different aspects in order to find the best solution for the case at hand. It is mainly done by architecture teams, often with several constraints imposed on time and budget spent. Due to this, only a very small fraction of the total number of options are considered, of which even fewer arrive to the stages of structural design, to finally be simulated and tested by the aforementioned programs. However, the structural component can heavily influence the outcome of a construction, not only with respect to the safety, but at several different levels, namely functionality, sustainability and economics. Thus, in recent years, several attempts have been made to come up with software aids which are more flexible (see [1] and [2]), with the intent to achieve a better balance in the design of buildings.

The motivation for this dissertation comes from the desire to create a piece of software that enables structural analysis to enter the scene in earlier design stages, namely conceptual design. The goal is to write a code that can perform both static and dynamic analysis in an expeditious manner; capturing the most essential components of the behavior of structures, while not being overly complex, in order to extend the space of possibilities for any given project, which could help in the efficient design of buildings.

Such a program should be able to give a good structural overview of a model with as little input data or effort from the user as possible, while being flexible enough to allow for most kinds of designs (like setbacks, or asymmetric buildings). It should make use of a few good simplifications based in the theory of structural behavior, leaving the more resource intensive and detailed studies for later stages of the project. It should also give the possibility to perform parametric studies, such as varying the position or cross-sections of certain elements within the structure.

1.2 Outline of the dissertation

The present document is divided in seven chapters:

In Chapter 2, The main outline on the approach to the structural behavior modelling is given, along with the main simplifications and assumptions taken. The method of static condensation is presented, together with the introduction of global degrees of freedom which are common for all sub-structures.

In Chapter 3, the three different types of sub-structures that are considered in this work are presented. A discussion of their modelling ensues, along with a brief explanation of some details relevant to the methods introduced in Chapter 2.

Chapter 4 presents a short theoretical overview behind static and dynamic analysis, and the method of Response Spectra Analysis for seismic actions, prescribed in the Eurocode 8 [3]. An outline of the calculation of stresses in linear elements is given as well.

In Chapter 5 the implementation of the program is presented, detailing the approach taken as well as the organization and structure of the program.

In Chapter 6, several tests are performed on different simple models in order to evaluate the data obtained with expected results, which are then used to discuss and ultimately justify some of the simplifications applied, and also the main choices taken when deciding the important aspects to be included in the modelling. Finally, an evaluation of the overall quality of the program is attempted by considering a more realistic case study.

In Chapter 7, the final conclusions are presented, as well as some ideas for future work and research in the area.

2 MODELS WITH 3 DEGREES OF FREEDOM PER STORY

2.1 Description of the model

One of the first simplifications done in order to reduce the amount of computation and detail in the analysis is the assumption that structures modelled by the program have 3 main degrees of freedom per story. These correspond to translations in both horizontal coordinates, and rotation in the horizontal plane (see Figure 2.1 below). This assumption is justified due to the fact that for the seismic analysis, only horizontal accelerations will be considered, and also the fact that the slabs in a building usually exhibit a diaphragm behavior, making them axially rigid:

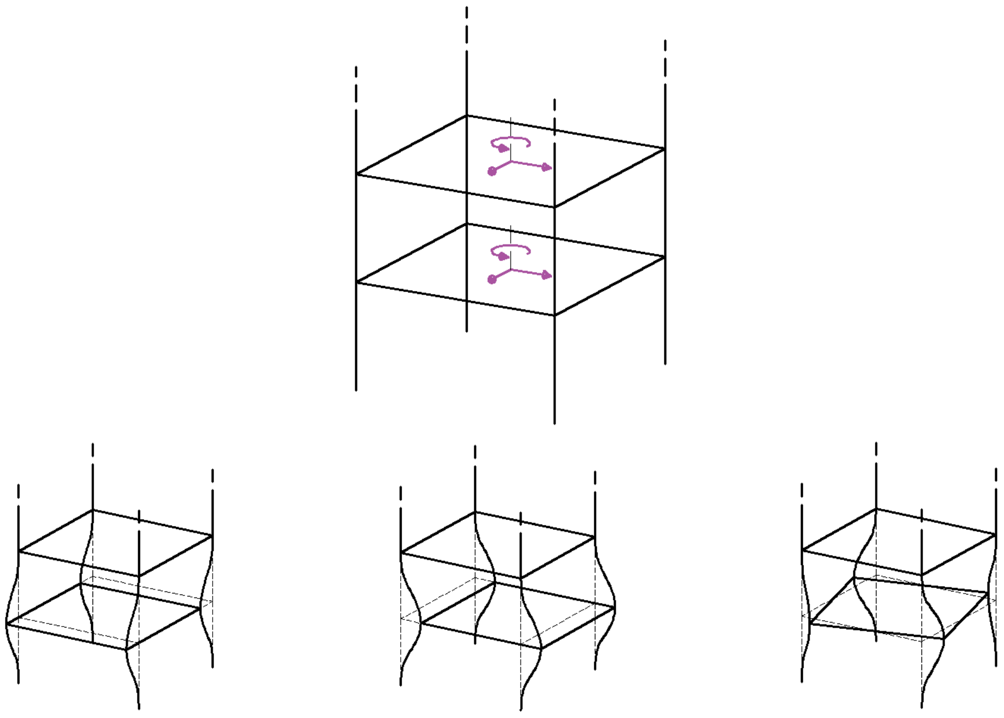


Figure 2.1 – The three main degrees of freedom per floor.

This observation, coupled with the wish for a simple way to define and model structures, led to the decision of considering three different main types of sub-structure – 2D frames, 3D frames, and core walls – which can be combined and used as building blocks to create the model of a building. An example of such a structure is the one presented in Figure 2.2, where the 3D frames are pictured in blue, 2D frames in green and core walls in orange:

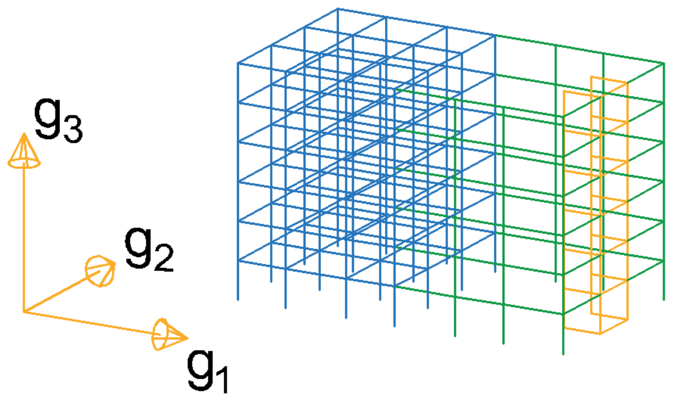


Figure 2.2 – Example of a model combining the different sub-structures.

Aiming to keep the structural model as simple as possible, these sub-structures are modelled using solely linear elements. There are no 2D elements, since the effect of the slabs is to be introduced as a load applied to the beams, and the walls can be thought of as being column elements with some changes, namely connectedness between several walls.

In order to study the whole structure and its degrees of freedom, it is also convenient to introduce a global frame of reference, with unit vectors $\overrightarrow{g_1}$, $\overrightarrow{g_2}$ and $\overrightarrow{g_3}$. The first two are horizontal and the third is vertical, as can be seen on Figure 2.2. They follow the usual cross product rule ($\overrightarrow{g_1} \times \overrightarrow{g_2} = \overrightarrow{g_3}$).

Hence, the three degrees of freedom per floor considered are identified as translations along axes $\overrightarrow{g_1}$ and $\overrightarrow{g_2}$; and rotation around axis $\overrightarrow{g_3}$. This ($\overrightarrow{g_1}, \overrightarrow{g_2}, \overrightarrow{g_3}$) frame of reference will be used throughout the present work as an absolute origin point from which all sub-structures will be defined.

2.2 Assembly of a model from its sub-structures

Having several sub-structures in a model leads to some difficulties for the study of their behavior because while it is very easy to analyze them separately, they are interconnected and thus interact with each other. This creates a need for finding a method to collect the information about each substructure (applied forces, constrained nodes and stiffnesses), and then abstract from these and perform an analysis on the structure as a whole.

2.2.1 GLOBAL, “LOCAL” AND “SLAVE” DEGREES OF FREEDOM

Following the definitions introduced in the previous Section, it would be helpful to find a way of analyzing the structure using only the three degrees of freedom per floor. Thus, the goal is to transform the static system of equations of each sub-structure into another one in which the unknowns are exactly the displacements along the global degrees of freedom, $\boldsymbol{q}_g$:

$$\boldsymbol{q}_g = \left.\begin{bmatrix} q_{x,1} \\ q_{y,1} \\ q_{\theta,1} \\ - \\ \vdots \\ - \\ q_{x,n_f} \\ q_{y,n_f} \\ q_{\theta,n_f} \end{bmatrix}\right\} \text{size} = 3 \times n_f$$

where n_f is equal to the number of floors. As previously mentioned, the subscripts x and y refer to the degrees of freedom associated with the two translations along $\overrightarrow{g_1}$ and $\overrightarrow{g_2}$, respectively. Subscript θ denotes rotation around global frame axis $\overrightarrow{g_3}$.

This should give, for each substructure, a system of equations of the following form:

$$\boldsymbol{K}_{gg}^{(i)}\,\boldsymbol{q}_g = \boldsymbol{f}_g^{(i)} \tag{2.1}$$

where $\boldsymbol{K}_{gg}^{(i)}$ and $\boldsymbol{f}_g^{(i)}$ are the stiffness matrix and force vector belonging to the i^{th} substructure, relative to the global degrees of freedom $\boldsymbol{q}_g$.

The first thing to notice is that, for any sub-structure, each degree of freedom considered can be one of two different types:

- Degrees of freedom which do not depend directly on the three degrees of freedom per floor (corresponding to vertical translations and rotations along both horizontal axes) and from now on will be referred to as "Local" DOF's, written as $\boldsymbol{q}_l$;
- Degrees of freedom that correspond to horizontal displacements at the level of the slabs (horizontal translations and rotation along the vertical axis), and thus depend directly on the three main degrees of freedom per story previously introduced. These will be called "Slave" DOF's (to denote their dependence on the global DOF's) and abbreviated as $\boldsymbol{q}_s$.

This distinction suggests that there exists a relation between the "Slave" degrees of freedom of every sub-structure and the global degrees of freedom of the model, which could be capitalized on in order to derive the behavior of each such sub-structure from the global behavior of the whole model.

The new system referred to in (2.1) will be achieved firstly by taking the original system of equations of the sub-structure and writing both $\boldsymbol{q}_s$ and $\boldsymbol{q}_l$ in terms of just $\boldsymbol{q}_s$ through a method called static condensation of the system of equations; and then by writing all these $\boldsymbol{q}_s$ in terms of $\boldsymbol{q}_g$, since as it was mentioned before, the former depend directly on latter.

2.2.2 STATIC CONDENSATION

Static condensation is a method used with systems of equations whereby some of the variables are written in terms of others, making it easier to solve equations which depend only on those specific variables. In this case, it would be helpful to have the static system of each sub-structure depend only on the “Slave” degrees of freedom.

The starting point is a set of static systems of equations, one for each of the sub-structures that make up the model, of the form:

$$\begin{cases} \boldsymbol{K}^{(1)}\,\boldsymbol{q}^{(1)} = \boldsymbol{f}^{(1)} \\ \boldsymbol{K}^{(2)}\,\boldsymbol{q}^{(2)} = \boldsymbol{f}^{(2)} \\ \quad\vdots \end{cases} \tag{2.2}$$

Let us consider a sub-structure and its degrees of freedom $\boldsymbol{q}^{(i)}$, which in the following expressions are referred to as just $\boldsymbol{q}$ for simplicity. Looking again at the distinction between “Slave” and “Local” degrees of freedom, it is clear that they can be reordered and partitioned in the following manner:

$$\boldsymbol{q}^{(i)} = \boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_s \\ - \\ \boldsymbol{q}_l \end{bmatrix} \tag{2.3}$$

With $\boldsymbol{q}_s$ standing for “Slave” degrees of freedom and $\boldsymbol{q}_l$ for “Local” degrees of freedom. Making the same partition on the static system of equations:

$$\begin{bmatrix} \boldsymbol{K}_{ss} & \boldsymbol{K}_{sl} \\ \boldsymbol{K}_{ls} & \boldsymbol{K}_{ll} \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_s \\ \boldsymbol{q}_l \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_s \\ \boldsymbol{f}_l \end{bmatrix} \Leftrightarrow \begin{cases} \boldsymbol{K}_{ss}\,\boldsymbol{q}_s + \boldsymbol{K}_{sl}\,\boldsymbol{q}_l = \boldsymbol{f}_s \\ \boldsymbol{K}_{ls}\,\boldsymbol{q}_s + \boldsymbol{K}_{ll}\,\boldsymbol{q}_l = \boldsymbol{f}_l \end{cases} \tag{2.4}$$

One can take the second line of (2.4) and write $\boldsymbol{q}_l$ in terms of $\boldsymbol{q}_s$

$$\boldsymbol{K}_{ls}\,\boldsymbol{q}_s + \boldsymbol{K}_{ll}\,\boldsymbol{q}_l = \boldsymbol{f}_l$$

$$\boldsymbol{q}_l = \boldsymbol{K}_{ll}^{-1}(\boldsymbol{f}_l - \boldsymbol{K}_{ls}\,\boldsymbol{q}_s) = \boldsymbol{K}_{ll}^{-1}\,\boldsymbol{f}_l - \boldsymbol{K}_{ll}^{-1}\,\boldsymbol{K}_{ls}\,\boldsymbol{q}_s \tag{2.5}$$

Then, substituting $\boldsymbol{q}_l$ in the first line of (2.4) and rearranging gives:

$$\boldsymbol{K}_{ss}\,\boldsymbol{q}_s + \boldsymbol{K}_{sl}\,(\boldsymbol{K}_{ll}^{-1}\,\boldsymbol{f}_l - \boldsymbol{K}_{ll}^{-1}\,\boldsymbol{K}_{ls}\,\boldsymbol{q}_s) = \boldsymbol{f}_s$$

$$(\boldsymbol{K}_{ss} - \boldsymbol{K}_{sl}\,\boldsymbol{K}_{ll}^{-1}\,\boldsymbol{K}_{ls})\,\boldsymbol{q}_s = \boldsymbol{f}_s - \boldsymbol{K}_{sl}\,\boldsymbol{K}_{ll}^{-1}\,\boldsymbol{f}_l$$

Using this last expression, it is then possible to define new stiffness matrix $\boldsymbol{K}_{ss}^{*}$ and force vector $\boldsymbol{f}_s^{*}$:

$$\boldsymbol{K}_{ss}^{*} = \boldsymbol{K}_{ss} - \boldsymbol{K}_{sl}\,\boldsymbol{K}_{ll}^{-1}\,\boldsymbol{K}_{ls} \tag{2.6}$$

$$\boldsymbol{f}_s^* = \boldsymbol{f}_s - \boldsymbol{K}_{sl}\,\boldsymbol{K}_{ll}^{-1}\,\boldsymbol{f}_l \tag{2.7}$$

Finally arriving at:

$$\boldsymbol{K}_{ss}^*\,\boldsymbol{q}_s = \boldsymbol{f}_s^* \tag{2.8}$$

which is the static system of equations rewritten only in terms of the “Slave” displacements, as intended.

2.2.3 TRANSFORMATION INTO GLOBAL COORDINATES

At this point, it would be helpful to find a way to write $\boldsymbol{q}_s$ in terms of $\boldsymbol{q}_g$ in matrix form. This is indeed possible because as previously mentioned, every entry of $\boldsymbol{q}_s$ is a displacement that depends directly on the three global degrees of freedom of its corresponding floor, and thus it can be written as a linear combination of the components of vector $\boldsymbol{q}_g$.

This means that for any sub-structure, its “Slave” displacements belonging to the j^{th} floor relate to the corresponding global displacements $\boldsymbol{q}_g^j$ by means of:

$$\boldsymbol{q}_s^j = \boldsymbol{A}_{sg}^j\,\boldsymbol{q}_g^j \tag{2.9}$$

where $\boldsymbol{A}_{sg}^j$ is a matrix with 3 columns and number of lines equal to the number of “Slave” degrees of freedom in the j^{th} floor of the sub-structure; and $\boldsymbol{q}_g^j$ is a vector containing the displacements along the three global degrees of freedom of the story:

$$\boldsymbol{q}_g^j = \begin{bmatrix} q_x^j \\ q_y^j \\ q_\theta^j \end{bmatrix}$$

After this, it is possible to create a matrix $\boldsymbol{A}_{sg}$ with a number of rows equal to the number of different “Slave” degrees of freedom in the sub-structure, where the transformation matrices for each story j, $\boldsymbol{A}_{sg}^j$, are assembled along the diagonal of $\boldsymbol{A}_{sg}$:

$$\boldsymbol{A}_{sg} = \left[\begin{array}{c|c|c|c} \boldsymbol{A}_{sg}^1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{A}_{sg}^2 & \cdots & \boldsymbol{0} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{A}_{sg}^n \end{array}\right] \tag{2.10}$$

with n being the number of floors in the whole structure.

This transformation matrix makes it then possible to write the relation between “Slave” and “Global” degrees of freedom simply as:

$$\boldsymbol{q}_s = \boldsymbol{A}_{sg}\, \boldsymbol{q}_g \tag{2.11}$$

Going back to equation (2.8), and applying the Principle of Virtual Work:

$$\boldsymbol{K}_{ss}^{*}\, \boldsymbol{q}_s = \boldsymbol{f}_s^{*}$$

$$\boldsymbol{K}_{ss}^{*}\, \boldsymbol{q}_s - \boldsymbol{f}_s^{*} = \boldsymbol{0}$$

$$(\boldsymbol{K}_{ss}^{*}\, \boldsymbol{q}_s - \boldsymbol{f}_s^{*}) \cdot \delta \boldsymbol{q}_s = 0, \qquad \forall \delta \boldsymbol{q}_s$$

Where “·” represents the inner (dot) product of vectors. Now, using the relation found between $\boldsymbol{q}_s$ and $\boldsymbol{q}_g$ it is possible to go further:

$$\boldsymbol{q}_s = \boldsymbol{A}_{sg}\, \boldsymbol{q}_g \qquad \Rightarrow \qquad \delta \boldsymbol{q}_s = \boldsymbol{A}_{sg}\, \delta \boldsymbol{q}_g$$

Substituting and rearranging the terms:

$$\left(\boldsymbol{K}_{ss}^{*}\, \boldsymbol{A}_{sg}\, \boldsymbol{q}_g - \boldsymbol{f}_s^{*}\right) \cdot \left(\boldsymbol{A}_{sg}\, \delta \boldsymbol{q}_g\right) = 0$$

$$\boldsymbol{A}_{sg}^{T}\left(\boldsymbol{K}_{ss}^{*}\, \boldsymbol{A}_{sg}\, \boldsymbol{q}_g - \boldsymbol{f}_s^{*}\right) \cdot \delta \boldsymbol{q}_g = 0, \qquad \forall \delta \boldsymbol{q}_g$$

$$\boldsymbol{A}_{sg}^{T}\left(\boldsymbol{K}_{ss}^{*}\, \boldsymbol{A}_{sg}\, \boldsymbol{q}_g - \boldsymbol{f}_s^{*}\right) = \boldsymbol{0}$$

$$\left(\boldsymbol{A}_{sg}^{T}\, \boldsymbol{K}_{ss}^{*}\, \boldsymbol{A}_{sg}\right) \boldsymbol{q}_g = \boldsymbol{A}_{sg}^{T}\, \boldsymbol{f}_s^{*}$$

In a similar fashion to what was done with equations (2.6) and (2.7), the global stiffness matrix, $\boldsymbol{K}_{gg}$ and the global forces vector, $\boldsymbol{f}_g$ are then defined as:

$$\boldsymbol{K}_{gg} = \boldsymbol{A}_{sg}^{T}\, \boldsymbol{K}_{ss}^{*}\, \boldsymbol{A}_{sg} \tag{2.12}$$

$$\boldsymbol{f}_g = \boldsymbol{A}_{sg}^{T}\, \boldsymbol{f}_s^{*} \tag{2.13}$$

Arriving as intended to the equation system (2.1), which is written in terms of $\boldsymbol{q}_g$ (and recovering the previous notation to make explicit the terms related to the i^{th} sub-structure):

$$\boldsymbol{K}_{gg}^{(i)}\, \boldsymbol{q}_g = \boldsymbol{f}_g^{(i)}$$

Since the global degrees of freedom are shared by all sub-structures, the contribution of each sub-structure to these global displacements can then be directly assembled into a global system of equations of the same form, thus giving the possibility for the structure to be studied as a whole. In order to assemble these contributions, it is simply needed to take the sum of the stiffnesses and forces over all n sub-structures:

$$\left(\sum_{i=1}^{n} \boldsymbol{K}_{gg}^{(i)}\right) \boldsymbol{q}_g = \sum_{i=1}^{n} \boldsymbol{f}_g^{(i)} \tag{2.14}$$

which is the global system of equations sought in order to study the model as a whole. In particular, by adding all the force vectors $\boldsymbol{f}_g^{(i)}$ there is no need to consider explicitly the transfer of internal forces between substructures.

3 SUB-STRUCTURES

In this chapter, the theoretical formulation for the modelling of all sub-structures is discussed, presenting the derivations for expressions that are used in the program to compute everything that is necessary for the analysis, as well as the assumptions and simplifications that were made in order to make the program accessible and not too computationally demanding.

As previously mentioned, the types of sub-structure considered are three: 2D frames, 3D frames and cores walls. The main components of all these substructures are linear elements connecting nodes, which results in some similarities between all of them. The process is roughly the same for all sub-structures, and goes as follows:

1. Definition of a reference frame for the sub-structure;
2. Numbering of the nodes and free displacements of the sub-structure;
3. Numbering of the linear elements connecting nodes, definition of their local coordinate frames and computation of their fixed end force vectors and stiffness matrices;
4. Assembly of the force vector and stiffness matrix of the whole sub-structure;
5. Static condensation of this system of equations;
6. Computation of the transformation matrix $\boldsymbol{A}_{sg}$ and the contributions to the static system of the whole structure, $\boldsymbol{K}_{gg}$ and $\boldsymbol{f}_g$, from equation (2.1).

3.1 2D Frames

3.1.1 MAIN ASSUMPTIONS AND REFERENCE FRAME

The 2D frames are considered as having only stiffness in the plane of the frame. Hence, there are only 3 degrees of freedom per node: rotation in the plane of the frame, and horizontal and vertical translations. Only the horizontal displacements contribute directly to the global degrees of freedom of the structure. It is assumed that the bottom level nodes are fixed for all displacements, and all the other nodes are free from constraints.

If N is the number of nodes in the structure, and N_f is the number of fixed displacements, the number of static degrees of freedom in the structure is:

$$n_{dof} = 3N - N_f$$

The number of degrees of freedom is then naturally the number of static equations and unknowns in the structure.

A new frame of reference for the 2D frame, $(\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3})$. is defined, with its origin at the starting point of the frame, and axes pointing in the following manner:

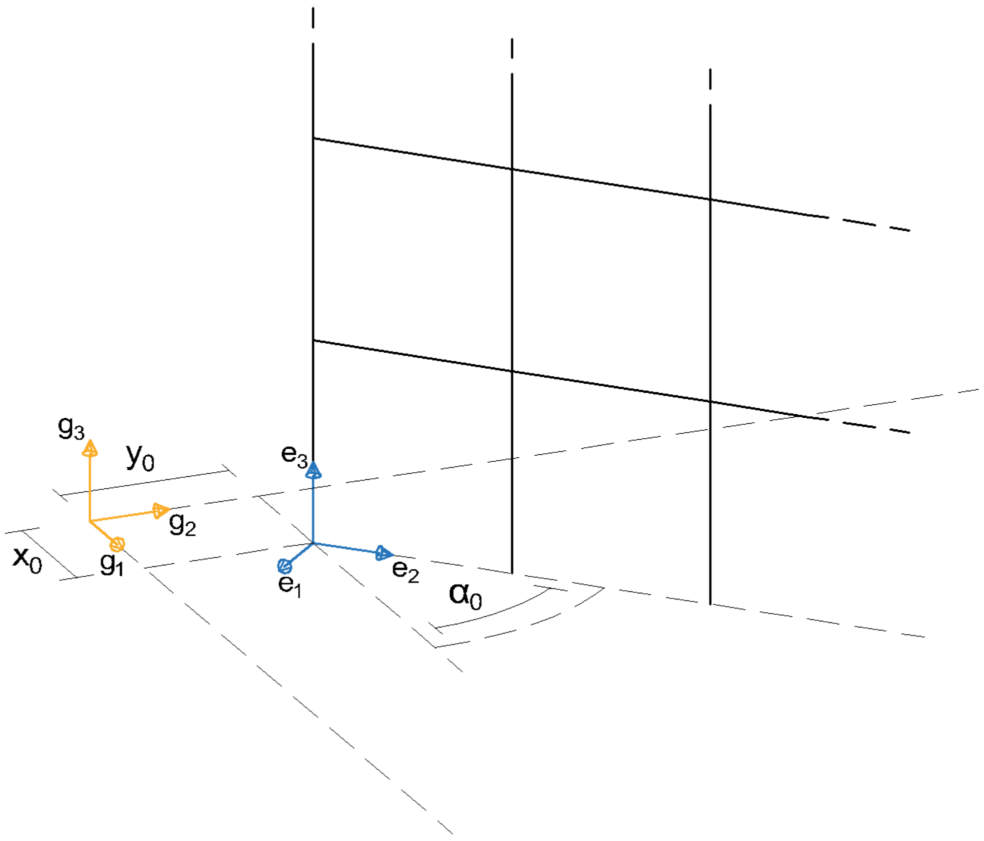


Figure 3.1 – Positioning of the 2D frame with respect to the global axes.

In the previous figure, global axes are in orange, and the 2D frame axes are in blue. These are located at the point $(x_0, y_0, 0)$ in global coordinates, with angle α_0 between $\overrightarrow{g_1}$ and $\overrightarrow{e_2}$.

This frame of reference will be used to analyze the 2D frame in order to simplify the calculations, since it is easier to think of the 2D frame as being contained in a 2D plane rather than as a structure having coordinates in 3D. This is also the frame of reference used for the displacements in the external forces vector.

3.1.2 LINEAR ELEMENTS OF 2D FRAMES

Each linear element can be thought of as having associated to it a local reference frame, $(\vec{l_1}, \vec{l_2}, \vec{l_3})$. The first axis lies along the direction of the element, the second is perpendicular to the plane of the frame in the same direction as $\vec{e_1}$, and the third is defined as the cross product of $\vec{l_1} \times \vec{l_2}$:

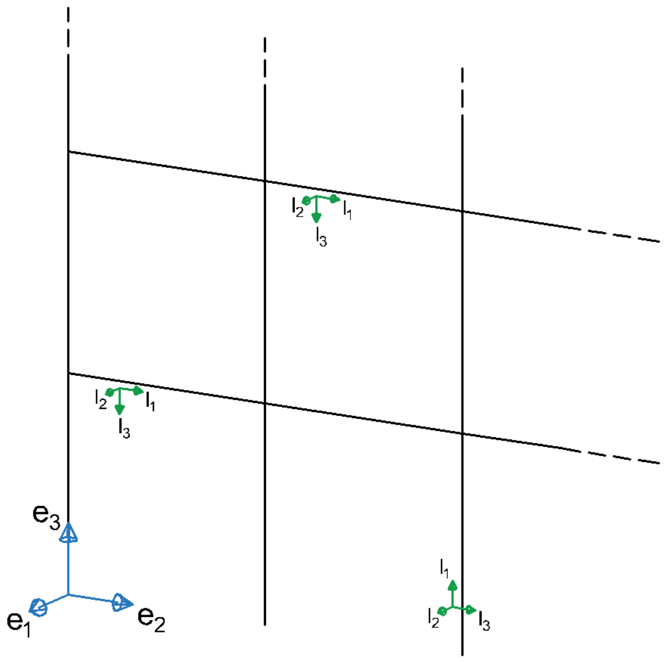


Figure 3.2 – Example of local frames of reference for three of the elements in a 2D frame.

Each element connects two nodes, which in turn have three degrees of freedom each, so for every element, its displacements acting on the nodes can be written with respect to the local coordinates and arranged in a six-by-one vector of the form:

$$\boldsymbol{q}_l^b = \begin{bmatrix} \boldsymbol{q}_l^i \\ \boldsymbol{q}_l^f \end{bmatrix} \tag{3.1}$$

Having for each node (see Figure 3.3):

$$\boldsymbol{q}_l^i = \begin{bmatrix} q_{l1}^i \\ q_{l2}^i \\ q_{l3}^i \end{bmatrix}; \qquad \boldsymbol{q}_l^f = \begin{bmatrix} q_{l1}^f \\ q_{l2}^f \\ q_{l3}^f \end{bmatrix} \tag{3.2}$$

Where the superscripts i and j refer to the starting and ending nodes of the element, respectively.

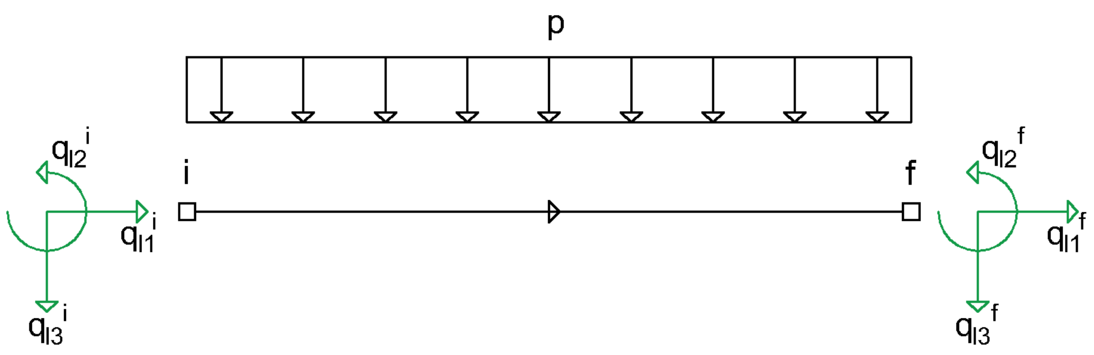


Figure 3.3 – Displacements in a linear element belonging to a 2D frame.

Looking at these local frames defined for each element, as presented in Figure 3.2, one can verify that it is possible to write them in terms of the coordinates of the 2D frame through a transformation matrix $\boldsymbol{A}$ such that:

$$\boldsymbol{q}_l = \boldsymbol{A}\,\boldsymbol{q}_e \tag{3.3}$$

For beams:

$$\begin{cases} q_{l1} = q_{e2} \\ q_{l2} = q_{e1} \\ q_{l3} = -q_{e3} \end{cases} \Leftrightarrow \boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

And for columns:

$$\begin{cases} q_{l1} = q_{e3} \\ q_{l2} = q_{e1} \\ q_{l3} = q_{e2} \end{cases} \Leftrightarrow \boldsymbol{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

These vectors can be written with respect to the 2D frame coordinates $(\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3})$ by defining a new transformation matrix $\boldsymbol{\Lambda}$ which contains two copies of the matrix $\boldsymbol{A}$ in its diagonal:

$$\begin{bmatrix} q_{l1}^{i} \\ q_{l2}^{i} \\ q_{l3}^{i} \\ q_{l1}^{f} \\ q_{l2}^{f} \\ q_{l3}^{f} \end{bmatrix} = \left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{A} \end{array}\right] \begin{bmatrix} q_{e1}^{i} \\ q_{e2}^{i} \\ q_{e3}^{i} \\ q_{e1}^{f} \\ q_{e2}^{f} \\ q_{e3}^{f} \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{q}_l^b = \boldsymbol{\Lambda}\,\boldsymbol{q}_e^b \tag{3.5}$$

The stiffness matrix for a linear element in terms of its local displacements, $\boldsymbol{K}_l^b$ is:

$$\boldsymbol{K}_l^b = \begin{bmatrix} EA/L & 0 & 0 & -EA/L & 0 & 0 \\ & 4EI/L & -6EI/L^2 & 0 & 2EI/L & 6EI/L^2 \\ & & 12EI/L^3 & 0 & -6EI/L^2 & -12EI/L^3 \\ & & & EA/L & 0 & 0 \\ & Sym. & & & 4EI/L & 6EI/L^2 \\ & & & & & 12EI/L^3 \end{bmatrix} \tag{3.6}$$

This matrix can be multiplied by the displacements to give the forces acting along all degrees of freedom of the element:

$$\boldsymbol{f}_l^b = \boldsymbol{K}_l^b\, \boldsymbol{q}_l^b \tag{3.7}$$

The stiffness matrix can then be written in terms of the 2D frame coordinates by resorting to this last equation, together with (3.5):

$$\left(\boldsymbol{K}_l^b\, \boldsymbol{q}_l^b - \boldsymbol{f}_l^b\right) \cdot \delta\boldsymbol{q}_l^b = 0, \qquad \forall \delta\boldsymbol{q}_l^b$$

$$\left(\boldsymbol{K}_l^b\, \boldsymbol{\Lambda}\, \boldsymbol{q}_e^b - \boldsymbol{f}_l^b\right) \cdot \left(\boldsymbol{\Lambda}\, \delta\boldsymbol{q}_e^b\right) = 0, \qquad \forall \delta\boldsymbol{q}_e^b$$

$$\boldsymbol{\Lambda}^T\left(\boldsymbol{K}_l^b \boldsymbol{\Lambda}\, \boldsymbol{q}_e^b - \boldsymbol{f}_l^b\right) \cdot \delta\boldsymbol{q}_e^b = 0$$

$$\left(\boldsymbol{\Lambda}^T\, \boldsymbol{K}_l^b\, \boldsymbol{\Lambda}\right) \boldsymbol{q}_e^b = \boldsymbol{\Lambda}^T\, \boldsymbol{f}_l^b$$

which gives

$$\boldsymbol{K}_e^b = \boldsymbol{\Lambda}^T\, \boldsymbol{K}_l^b\, \boldsymbol{\Lambda} \tag{3.8}$$

$$\boldsymbol{f}_e^b = \boldsymbol{\Lambda}^T\, \boldsymbol{f}_l^b \tag{3.9}$$

After calculating the stiffness matrix of each element in the global coordinates, the stiffness matrix of the 2D frame, $\boldsymbol{K}_f$, can be assembled:

$$\boldsymbol{K}_f = \bigwedge_{b=1}^{\boldsymbol{n_b}} \boldsymbol{K}_e^b \tag{3.10}$$

Resulting in a matrix of size $n_{dof} \times n_{dof}$, with n_b being the number of linear elements, and n_{dof} the number of static degrees of freedom of the frame.

Calculating the fixed end actions in the frame coordinates is very similar to the assembly of the stiffness matrix. For any beam with constant load p acting along $\overrightarrow{l_3}$ (see Figure 3.3), the fixed end actions of the element are given by the following vector:

$$\boldsymbol{f}_{l,fix}^{b} = \begin{bmatrix} f_{l1}^{i} \\ f_{l2}^{i} \\ f_{l3}^{i} \\ f_{l1}^{f} \\ f_{l2}^{f} \\ f_{l3}^{f} \end{bmatrix} = \begin{bmatrix} 0 \\ pL^2/12 \\ -pL/2 \\ 0 \\ -pL^2/12 \\ -pL/2 \end{bmatrix} \tag{3.11}$$

Where f_{lk}^{i} and f_{lk}^{f} are the fixed end forces or moments applied along the k^{th} local displacement of the starting and ending nodes of the element, respectively.

This vector can then be converted to global coordinates, $\boldsymbol{f}_{e,fix}^{b}$, making use of equation (3.9):

$$\boldsymbol{f}_{e,fix}^{b} = \boldsymbol{\Lambda}^{T}\, \boldsymbol{f}_{l,fix}^{b} \tag{3.12}$$

and assembled into the fixed forces vector of the structure, with size n_{dof}:

$$\boldsymbol{f}_{\boldsymbol{fix}} = \mathop{\mathrm{A}}_{b=1}^{\boldsymbol{n_b}} \boldsymbol{f}_{e,fix}^{b} \tag{3.13}$$

With n_b being the number of linear elements, and n_{dof} the number of static degrees of freedom of the frame once again.

The static system of equations of the structure is then defined as:

$$\boldsymbol{K}_{\boldsymbol{f}}\, \boldsymbol{q}_{\boldsymbol{f}} = \boldsymbol{f}_{\boldsymbol{f}} \tag{3.14}$$

with

$$\boldsymbol{f}_{\boldsymbol{f}} = \boldsymbol{f}_{\boldsymbol{ext}} - \boldsymbol{f}_{\boldsymbol{fix}} \tag{3.15}$$

After this, the matrix $\boldsymbol{K}_f$ and the vector $\boldsymbol{f}_f$ are ready for static condensation and assembly into the global model.

3.1.3 STATIC CONDENSATION OF 2D FRAMES

In the case of 2D frames only the horizontal displacements depend directly on the global displacements, making them the only ones to be considered as “Slave” displacements (shown in red in the following figure):

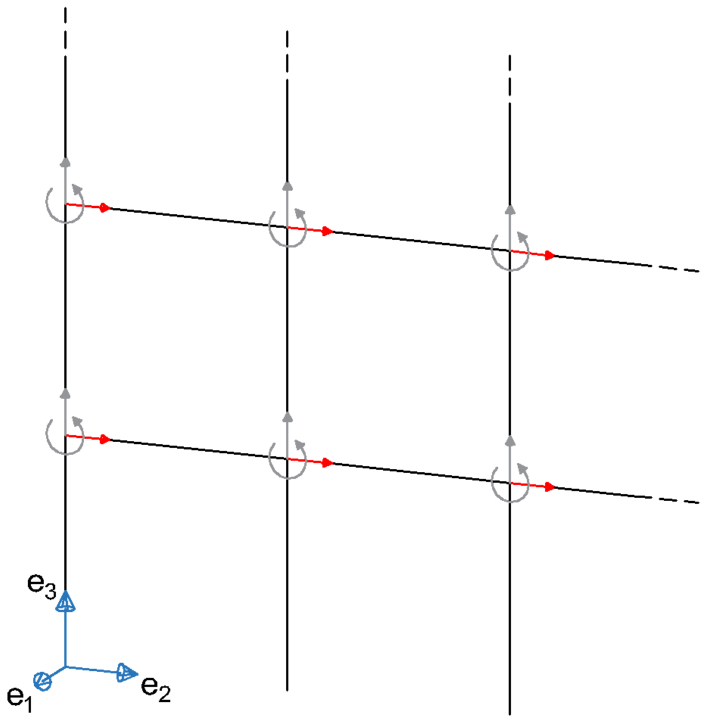


Figure 3.4 – Degrees of freedom in a 2D frame.

This means a partition can be made on the degrees of freedom of the frame, $\boldsymbol{q}_f$, as presented in equation (2.3):

$$\boldsymbol{q}_f = \begin{bmatrix} \boldsymbol{q}_s \\ - \\ \boldsymbol{q}_i \end{bmatrix}$$

Static condensation of the system can then be performed, as presented in Section 2.2.2, giving a new system of the type of equation (2.8):

$$\boldsymbol{K}_{ss}^{*}\, \boldsymbol{q}_s = \boldsymbol{f}_s^{*}$$

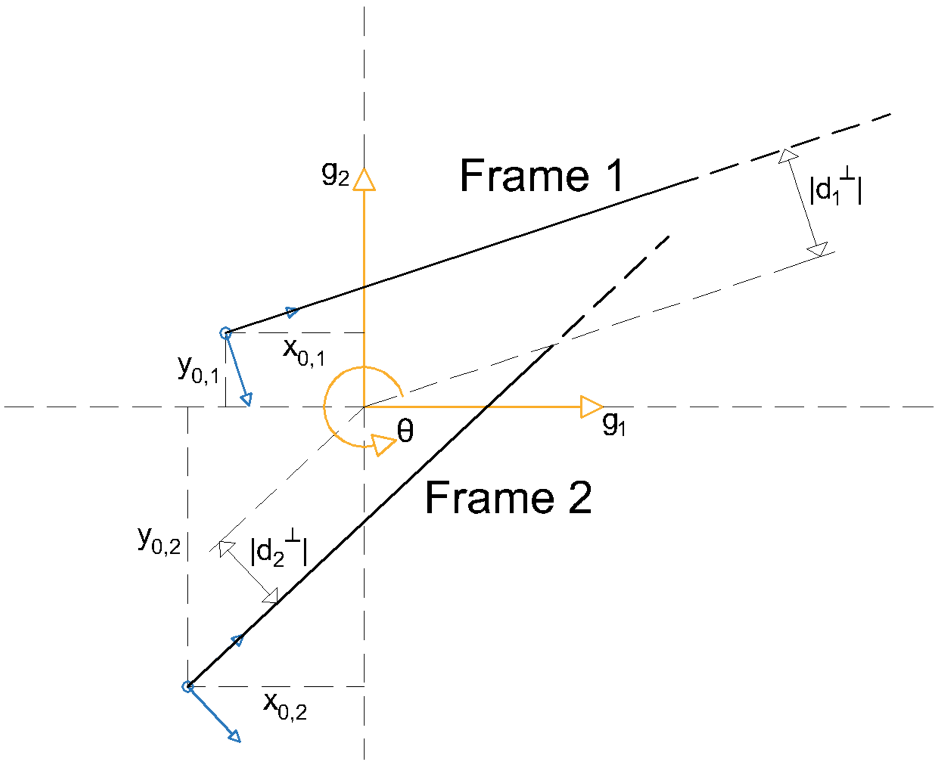
Then, every q_s^i can be written as:

$$q_s^i = [\cos\alpha_0 \quad \sin\alpha_0 \quad d^{\perp}] \begin{bmatrix} q_x^j \\ q_y^j \\ q_\theta^j \end{bmatrix} \tag{3.16}$$

In the previous equation, the subscript j refers to the floor in which the i^{th} degree of freedom is located. α_0 is the angle between direction x (global vector $\overrightarrow{g_1}$) and the direction of the frame $\overrightarrow{e_2}$ (see Figure 3.1); and $d^{\perp}$ is given by the following expression:

$$d^{\perp} = (\overrightarrow{g_3} \times \vec{x}) \cdot \overrightarrow{e_2} = \left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \begin{bmatrix} d_x \\ d_y \\ 0 \end{bmatrix} \right) \cdot \begin{bmatrix} \cos \alpha_0 \\ \sin \alpha_0 \\ 0 \end{bmatrix} = d_x \, sin \, \alpha_0 - d_y \, cos \, \alpha_0 \qquad (3.17)$$

Where $\vec{x}$ is the distance vector from the node in question to the origin of the global frame of reference, and the symbols "×" and "·" represent respectively the vector cross and dot products. In fact, $d^{\perp}$ is the signed distance of the frame from the origin measured in the direction orthogonal to the plane of the 2D frame (hence the notation used to represent it). The absolute value of this distance does not suffice, since in this case all rotations would be counted as translations in the direction of the frame. Thus, it is given a positive sign if $\overrightarrow{g_3} \times \vec{x}$ makes an angle of less than 90 degrees with the direction of $\overrightarrow{e_2}$. An example to illustrate this fact is given below:



It is also easy to notice that the value of $d^{\perp}$ depends only on the 2D frame, being the same for all nodes in it. Thus, the values for the distance of the node in equation (3.17) d_x and d_y can be exchanged for x_0 and y_0, which is the position of the origin of the 2D frame.

From here, the floor matrices $\boldsymbol{A}_{sg}^{j}$ referred to in Section 2.2.3 are assembled by gathering all slave displacements on each floor j:

$$\boldsymbol{A}_{sg}^{j} = \begin{bmatrix} \cos\alpha & \sin\alpha & d^{\perp} \\ & \vdots & \\ \cos\alpha & \sin\alpha & d^{\perp} \end{bmatrix}$$

Since all the “Slave” horizontal displacements in each floor are equal, all the rows in this matrix are identical.

Finally, a matrix $\boldsymbol{A}_{sg}$ relating “Slave” and “Global” displacements can be created, resorting to equation (2.10), arriving finally to the contributions from the 2D frame substructure calculated using equations (2.12) and (2.13). These can be added with the contributions of the other substructures in order to analyze the model as a whole, as discussed in the end of Section 2.2.3.

3.2 3D Frames

The modelling of 3D frames is very similar to the 2D case, the main difference being that every node now has six degrees of freedom, which correspond to three translations and three rotations. Of these six degrees, three contribute directly to the global degrees of freedom of the model: translations along both horizontal directions, and rotation in the horizontal plane.

If N is the number of nodes in the 3D frame, and N_f is the number of fixed displacements, the number of static degrees of freedom in the structure is:

$$n_{dof} = 6N - N_f$$

Once again, this is the number of static equations, which equals the size of both the frame stiffness matrix and the force vector.

From here, just as before, an orthonormal reference frame $(\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3})$ is introduced with coordinates $(x_0, y_0, 0)$ in the global coordinates, and making an angle α_0 between $\overrightarrow{g_1}$ and $\overrightarrow{e_1}$:

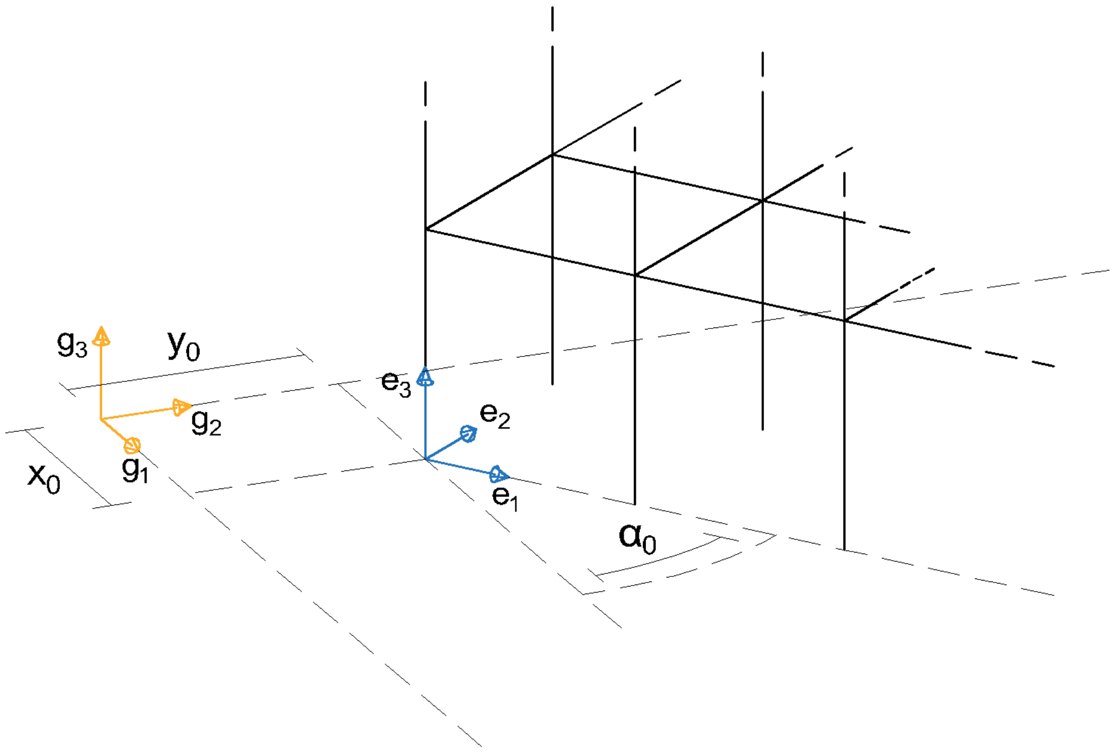


Figure 3.5 – Positioning of the frame of reference for a 3D frame.

This way, any set of nodal displacements can be written in a vector according to $(\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3})$:

$$\boldsymbol{q}_e = \begin{bmatrix} q_{e1} \\ q_{e2} \\ q_{e3} \\ q_{e4} \\ q_{e5} \\ q_{e6} \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{translations in } (\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3}) \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{rotations around } (\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3}) \end{matrix} \tag{3.18}$$

As in the 2D case, it is assumed that all the bottom floor nodes are fixed for all displacements, and all the others are free.

3.2.1 LINEAR ELEMENTS OF 3D FRAMES

Each linear element can be associated with a local reference frame $(\overrightarrow{l_1}, \overrightarrow{l_2}, \overrightarrow{l_3})$ defined as follows:

- If the element is vertical, $(\overrightarrow{l_1}, \overrightarrow{l_2}, \overrightarrow{l_3})$ are aligned with $(\overrightarrow{e_3}, \overrightarrow{e_1}, \overrightarrow{e_2})$, respectively;

- If the element has any non-zero horizontal component, $\overrightarrow{l_1}$ lies in the direction of the element, $\overrightarrow{l_2}$ has no vertical component, and $\overrightarrow{l_3}$ points downwards.

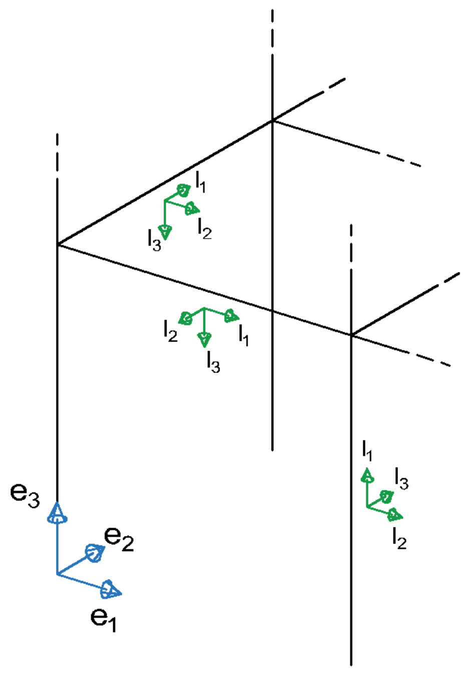


Figure 3.6 – Example of local reference frames for three elements in a 3D frame.

Thus, any set of displacements acting on a beam can be presented as a 12-by-1 vector, in local coordinates just as before:

$$\boldsymbol{q}_l^b = \begin{bmatrix} \boldsymbol{q}_l^i \\ \boldsymbol{q}_l^f \end{bmatrix} \tag{3.19}$$

Where the superscripts i and f refer to the start and end nodes of the beam.

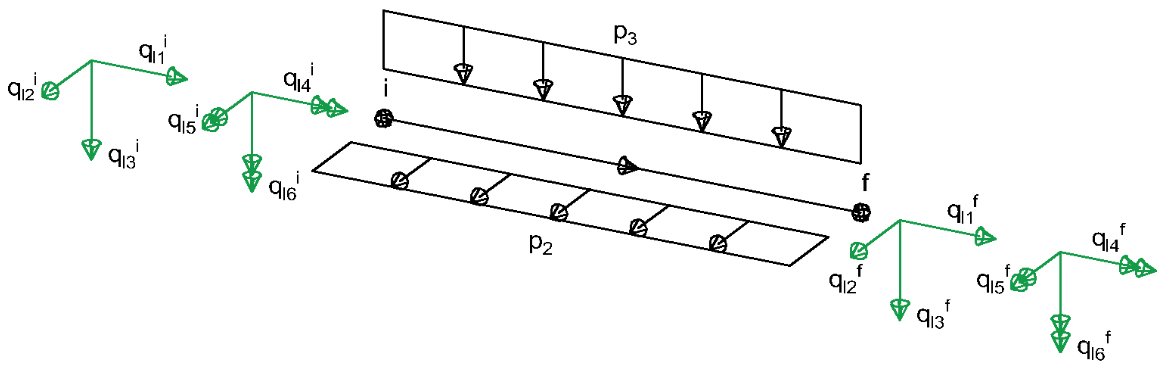


Figure 3.7 – Displacements in a beam element belonging to a 3D frame.

Each local frame $(\overrightarrow{l_1}, \overrightarrow{l_2}, \overrightarrow{l_3})$ is associated with a cosine matrix $\boldsymbol{A}$ which can be used to get the local coordinates of a vector written with respect to $(\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3})$, the coordinate system of the 3D frame:

$$\boldsymbol{A}_{ij} = \cos(\vec{l_i}, \vec{e_j})$$

In a similar fashion to the 2D case, these can then be written in terms of $(\vec{e_1}, \vec{e_2}, \vec{e_3})$ by means of a transformation matrix $\boldsymbol{\Lambda}$, which is now a 12-by-12 matrix with four copies of $\boldsymbol{A}$ along its diagonal:

$$\begin{bmatrix} q^i_{l1} \\ q^i_{l2} \\ q^i_{l3} \\ q^i_{l4} \\ q^i_{l5} \\ q^i_{l6} \\ q^f_{l1} \\ q^f_{l2} \\ q^f_{l3} \\ q^f_{l4} \\ q^f_{l5} \\ q^f_{l6} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} q^i_{l1} \\ q^i_{l2} \\ q^i_{l3} \\ q^i_{l4} \\ q^i_{l5} \\ q^i_{l6} \\ q^f_{l1} \\ q^f_{l2} \\ q^f_{l3} \\ q^f_{l4} \\ q^f_{l5} \\ q^f_{l6} \end{bmatrix}$$

$$\boldsymbol{q}^b_l = \boldsymbol{\Lambda}\, \boldsymbol{q}^b_e \tag{3.20}$$

The stiffness matrix of the 3D element in local coordinates, $\boldsymbol{K}^b_l$ is:

$$\begin{bmatrix}
{}^{EA}/_{L} & 0 & 0 & 0 & 0 & 0 & -{}^{EA}/_{L} & 0 & 0 & 0 & 0 & 0 \\
 & {}^{12EI}/_{L^3} & 0 & 0 & 0 & {}^{6EI}/_{L^2} & 0 & -{}^{12EI}/_{L^3} & 0 & 0 & 0 & {}^{6EI}/_{L^2} \\
 & & {}^{12EI}/_{L^3} & 0 & -{}^{6EI}/_{L^2} & 0 & 0 & 0 & -{}^{12EI}/_{L^3} & 0 & {}^{6EI}/_{L^2} & 0 \\
 & & & {}^{GJ}/_{L} & 0 & 0 & 0 & 0 & 0 & -{}^{GJ}/_{L} & 0 & 0 \\
 & & & & {}^{4EI}/_{L} & 0 & 0 & 0 & -{}^{6EI}/_{L^2} & 0 & {}^{2EI}/_{L} & 0 \\
 & & & & & {}^{4EI}/_{L} & 0 & -{}^{6EI}/_{L^2} & 0 & 0 & 0 & {}^{2EI}/_{L} \\
 & & & & & & {}^{EA}/_{L} & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & {}^{12EI}/_{L^3} & 0 & 0 & 0 & -{}^{6EI}/_{L^2} \\
 & & & & & & & & {}^{12EI}/_{L^3} & 0 & {}^{6EI}/_{L^2} & 0 \\
 & & \text{Sym.} & & & & & & & {}^{GJ}/_{L} & 0 & 0 \\
 & & & & & & & & & & {}^{4EI}/_{L} & 0 \\
 & & & & & & & & & & & {}^{4EI}/_{L}
\end{bmatrix}$$

Just as in the 2D case with equation (3.7), this matrix relates the displacements of the element with the forces acting on it:

$$\boldsymbol{f}^b_l = \boldsymbol{K}^b_l\, \boldsymbol{q}^b_l \tag{3.21}$$

Using the same reasoning for the 2D frames, one can change it to $(\vec{e_1}, \vec{e_2}, \vec{e_3})$ coordinates making use of the transformation matrix $\boldsymbol{\Lambda}$:

$$\boldsymbol{K}_e^b = \boldsymbol{\Lambda}^T \, \boldsymbol{K}_l^b \, \boldsymbol{\Lambda} \tag{3.22}$$

$$\boldsymbol{f}_e^b = \boldsymbol{\Lambda}^T \, \boldsymbol{f}_l^b \tag{3.23}$$

Each entry is then assembled in the corresponding row and column of the stiffness matrix of the frame, $\boldsymbol{K}_f$, much in the same fashion as done before.

The fixed end actions of a 3D element under a load of p_2 and p_3 along local directions $\overrightarrow{e_2}$ and $\overrightarrow{e_3}$ respectively, can be written as:

$$\boldsymbol{f}_{l,fix}^b = \begin{bmatrix} f_{fix}^i \\ f_{fix}^f \end{bmatrix}_{local} = \begin{bmatrix} 0 & -\frac{p_2 L}{2} & -\frac{p_3 L}{2} & 0 & \frac{p_3 L^2}{12} & -\frac{p_2 L^2}{12} & 0 & -\frac{p_2 L}{2} & -\frac{p_3 L}{2} & 0 & -\frac{p_3 L^2}{12} & \frac{p_2 L^2}{12} \end{bmatrix}^T$$

These are then written in global coordinates by multiplying the vector by the transformation matrix $\boldsymbol{\Lambda}^T$, and assembled in the corresponding rows of the vector of fixed end actions of the structure, $\boldsymbol{f_{fix}}$.

Once this is done, one has arrived at the static system of equations

$$\boldsymbol{K}_f \, \boldsymbol{q}_f = \boldsymbol{f}_f \tag{3.24}$$

with

$$\boldsymbol{f}_f = \boldsymbol{f_{ext}} - \boldsymbol{f_{fix}} \tag{3.25}$$

The matrix $\boldsymbol{K}_f$ and vector $\boldsymbol{f}_f$ are then ready for static condensation and further analysis.

3.2.2 STATIC CONDENSATION OF 3D FRAMES

Of all displacements in 3D frames, only horizontal translations and rotations along the vertical axis depend directly on the 3 global displacements per story. This means that only those displacements will be considered “Slave” degrees of freedom (pictured in red in the following picture):

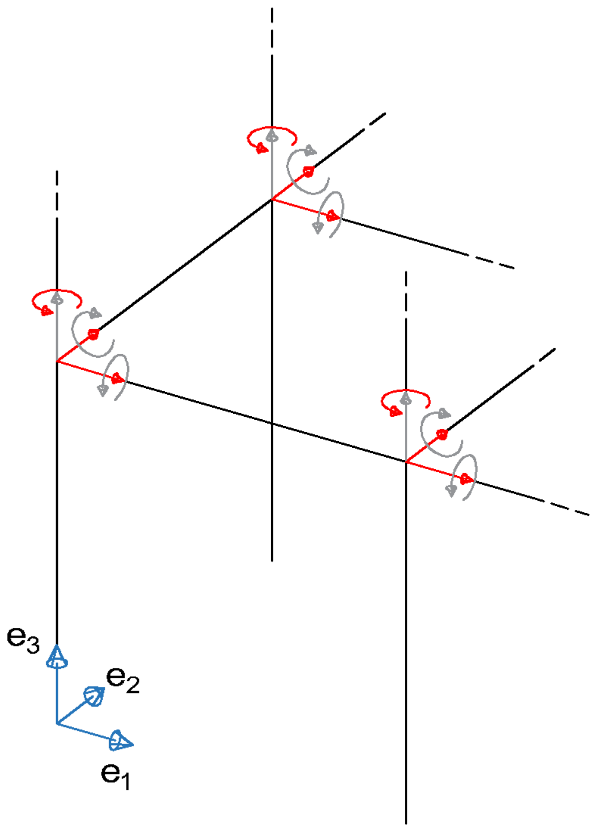


Figure 3.8 – Degrees of freedom in a 3D frame.

The unknowns of the static system of equations (3.21) can be partitioned as presented in equation (2.3):

$$\boldsymbol{q}_f = \begin{bmatrix} \boldsymbol{q}_s \\ - \\ \boldsymbol{q}_l \end{bmatrix}$$

Following this, the static condensation of the system of equations is performed according to Section 2.2.2 and a new system of the form of equation (2.8) is retrieved:

$$\boldsymbol{K}_{ss}^{*}\,\boldsymbol{q}_s = \boldsymbol{f}_s^{*}$$

This system describes the behavior of structure using only the slave displacements, $\boldsymbol{q}_s$. These, in turn, have the property that they can be written with respect to $\boldsymbol{q}_g$.This means that for any set of slave displacements acting on a node, $\boldsymbol{q}_s^i$ the following relation holds:

$$\boldsymbol{q}_s^i = \begin{bmatrix} q_{e1}^i \\ q_{e2}^i \\ q_{e6}^i \end{bmatrix} = \begin{bmatrix} \cos\alpha_0 & \sin\alpha_0 & d_1^\perp \\ -\sin\alpha_0 & \cos\alpha_0 & d_2^\perp \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_x^j \\ q_y^j \\ q_\theta^j \end{bmatrix}$$

With α_0 being the angle presented in Figure 3.5 and $d_n^\perp$ being given by the following expressions, similarly to (3.17):

$$\begin{cases} d_1^\perp = (\overrightarrow{g_3} \times \vec{x}) \cdot \overrightarrow{e_1} \\ d_2^\perp = (\overrightarrow{g_3} \times \vec{x}) \cdot \overrightarrow{e_2} \end{cases}$$

Again, $\vec{x}$ is a vector with the position of the node in question. Now, unlike for the 2D frames, $d_n^\perp$ doesn't depend only on the 3D frame it belongs to anymore and may change from node to node within the same 3D frame.

Using the same reasoning as in the 2D case, the matrices $\boldsymbol{A}_{sg}^j$ corresponding to each floor j can be assembled, which in turn are assembled into $\boldsymbol{A}_{sg}$, the matrix that relates the "Slave" DOF's and the global DOF's of the whole model, giving another instance of equation (2.11):

$$\boldsymbol{q}_s = \boldsymbol{A}_{sg}\,\boldsymbol{q}_g$$

And further, as described in Section 2.2.3, with equations (2.12) and (2.13):

$$\boldsymbol{K}_{gg} = \boldsymbol{A}_{sg}^T\,\boldsymbol{K}_{ss}^*\,\boldsymbol{A}_{sg}$$

$$\boldsymbol{f}_g = \boldsymbol{A}_{sg}^T\,\boldsymbol{f}_s^*$$

These give the contribution of the 3D frame to the whole model and can be summed respectively with the global stiffness matrices and forces vectors of other substructures to give finally the system for the whole model.

3.3 Core walls

The modelling of core walls is done with the help of a few simplifications. Individual walls are thought of as beam elements working only on a plane, following the Euler-Bernoulli assumption, which states that cross-sections will remain plane and perpendicular to the bending line.

Just like for the previous two types of sub-structures, a reference frame $(\vec{e_1}, \vec{e_2}, \vec{e_3})$ is defined:

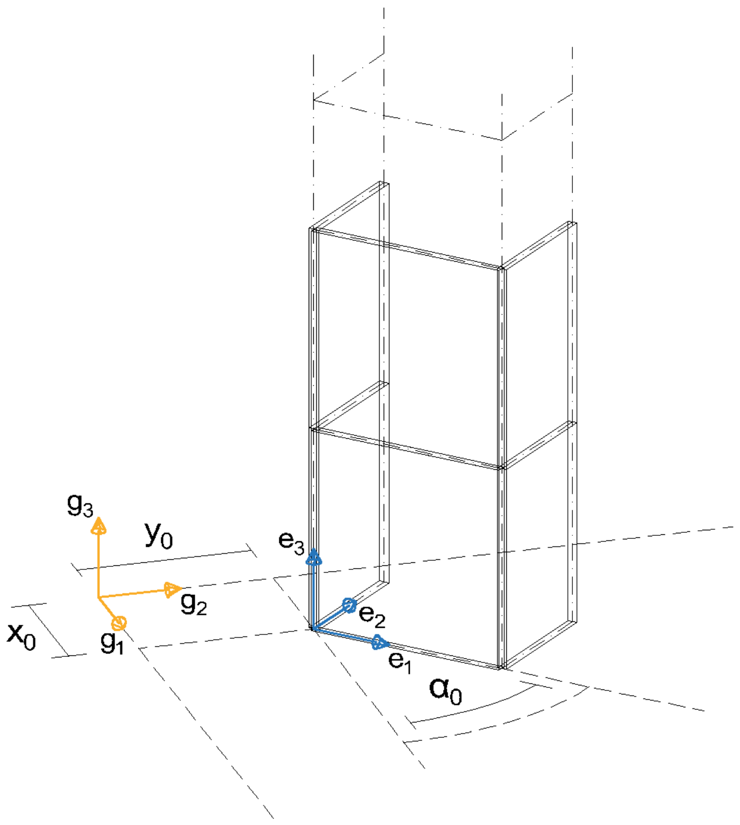


Figure 3.9 – Positioning of a core wall.

As usual, the frame of reference $(\vec{e_1}, \vec{e_2}, \vec{e_3})$ is defined by its position $(x_0, y_0, 0)$ and angle α_0 relative to the global frame of reference introduced at the start.

Two types of walls were considered: regular and connected walls. The two will now be presented in more detail.

3.3.1 REGULAR WALLS

In the present idealization, regular walls are thought of as being a linear element which connects to four nodes, and with its displacements being directly dependent of the displacements of these nodes.

Since the only displacements considered at the nodes of the walls are translations, displacements at a specific node can then be written in this reference frame as a vector with three coordinates $\boldsymbol{q}_e^n$, and given that each wall connects four nodes, its nodal displacements are expressed as a 12-by-1 vector, $\boldsymbol{q}_e^w$:

$$\boldsymbol{q}_e^n = \begin{bmatrix} q_{e1} \\ q_{e2} \\ q_{e3} \end{bmatrix} \quad ; \quad \boldsymbol{q}_e^w = \begin{bmatrix} \boldsymbol{q}_e^{n1} \\ \boldsymbol{q}_e^{n2} \\ \boldsymbol{q}_e^{n3} \\ \boldsymbol{q}_e^{n4} \end{bmatrix}$$

In order to think of the wall as a linear element, it is convenient to consider an orthonormal local reference frame $(\vec{l_1}, \vec{l_2}, \vec{l_3})$, similar to the one used before for linear elements (with vector $\vec{l_1}$ going from nodes 1 and 2 to nodes 3 and 4, and $\vec{l_3}$ in the direction of node 1 to node 2). This local frame is show in green in the following figure:

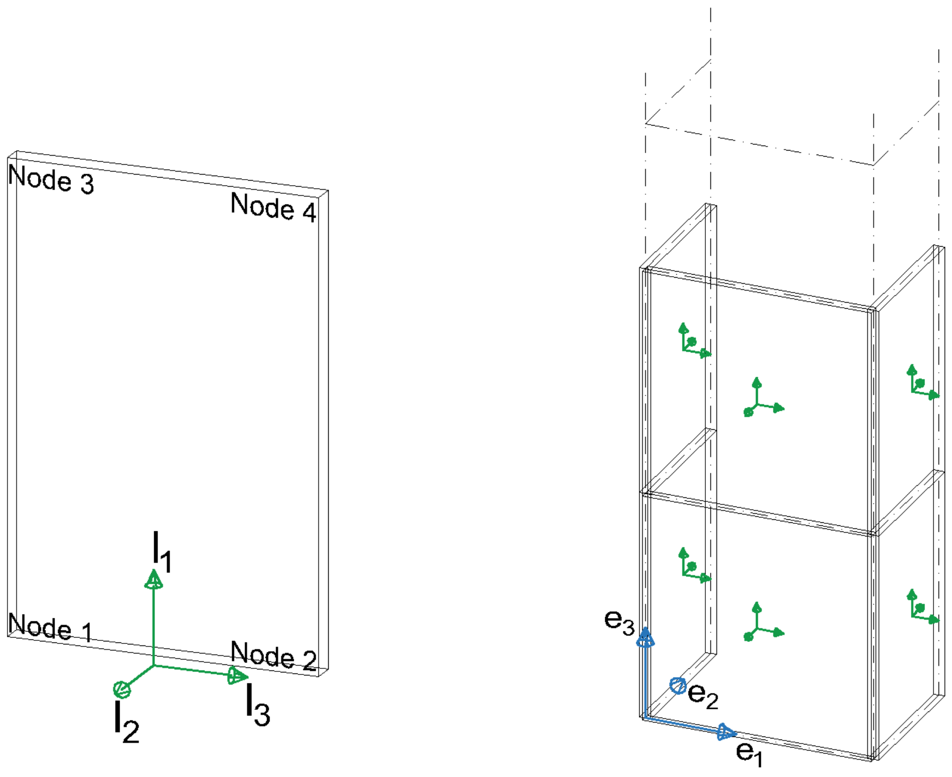


Figure 3.10 – Examples of the numbering of nodes and local reference frame for each wall.

As well as before, each local reference frame comes associated with an orthogonal matrix $\boldsymbol{A}$ which takes vectors in the $(\vec{e_1}, \vec{e_2}, \vec{e_3})$ frame and outputs them in the local frame:

$$\boldsymbol{q}_l^n = \boldsymbol{A}\, \boldsymbol{q}_e^n \tag{3.26}$$

Creating a matrix $\boldsymbol{\Lambda}$ with four copies of $\boldsymbol{A}$ down its main diagonal, it is possible to write the nodal displacements vector in terms of $(\vec{l_1}, \vec{l_2}, \vec{l_3})$ as well:

$$\boldsymbol{q}_l^w = \boldsymbol{\Lambda}\, \boldsymbol{q}_e^w \tag{3.27}$$

Using this newly defined local reference frame, the displacements of the wall can be considered as those appearing in a linear element, $\boldsymbol{q}_p$, and are then expressed according to $(\overrightarrow{l_1}, \overrightarrow{l_2}, \overrightarrow{l_3})$, in a similar way to the 2D frame element, as:

$$\boldsymbol{q}_p = \begin{bmatrix} q_{p1} \\ q_{p2} \\ q_{p3} \\ q_{p4} \\ q_{p5} \\ q_{p6} \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{displacements at the bottom of the wall} \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{displacements at the top of the wall} \end{matrix}$$

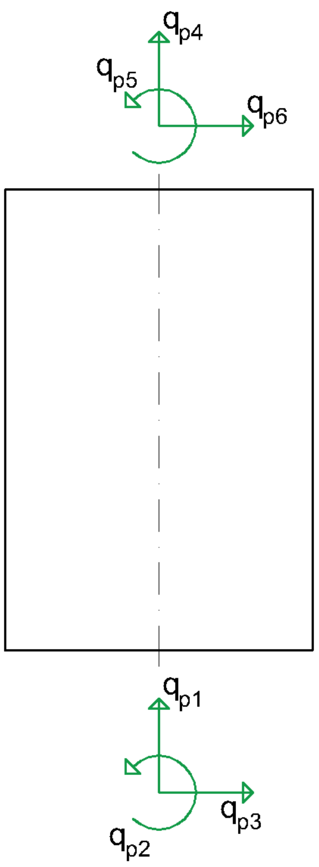


Figure 3.11 – The numbering of the displacements of a wall as a linear element.

This allows then to write a stiffness matrix for a wall which is very similar to that of a linear element belonging to a 2D frame (see Section 3.1.2).

$$\boldsymbol{K}_p = \begin{bmatrix} {EA}/{L} & 0 & 0 & -{EA}/{L} & 0 & 0 \\ & {4EI}/{L} & -{6EI}/{L^2} & 0 & {2EI}/{L} & {6EI}/{L^2} \\ & & {12EI}/{L^3} & 0 & -{6EI}/{L^2} & -{12EI}/{L^3} \\ & & & {EA}/{L} & 0 & 0 \\ & Sym. & & & {4EI}/{L} & {6EI}/{L^2} \\ & & & & & {12EI}/{L^3} \end{bmatrix} \qquad (3.28)$$

The stiffness matrix gives the forces acting along the degrees of freedom of the wall, when subject to a displacement vector $\boldsymbol{q}_p$:

$$\boldsymbol{f}_p = \boldsymbol{K}_p\, \boldsymbol{q}_p \tag{3.29}$$

It is possible to write the displacements $\boldsymbol{q}_p$ in terms of the nodal displacements vector, $\boldsymbol{q}_l^n$:

$$\begin{cases} q_{p1} = \dfrac{q_{l1}^{n1} + q_{l1}^{n2}}{2} \\ q_{p2} = \dfrac{-q_{l1}^{n1} + q_{l1}^{n2}}{b} \\ q_{p3} = \dfrac{q_{l3}^{n1} + q_{l3}^{n2}}{2} \end{cases} ; \qquad \begin{cases} q_{p4} = \dfrac{q_{l1}^{n3} + q_{l1}^{n4}}{2} \\ q_{p5} = \dfrac{-q_{l1}^{n3} + q_{l1}^{n4}}{b} \\ q_{p6} = \dfrac{q_{l3}^{n3} + q_{l3}^{n4}}{2} \end{cases} \tag{3.30}$$

where b is the width of the wall; and q_{lj}^{ni} is the displacement at the i^{th} node of the wall, along the direction of local reference frame vector $\vec{l}_j$:

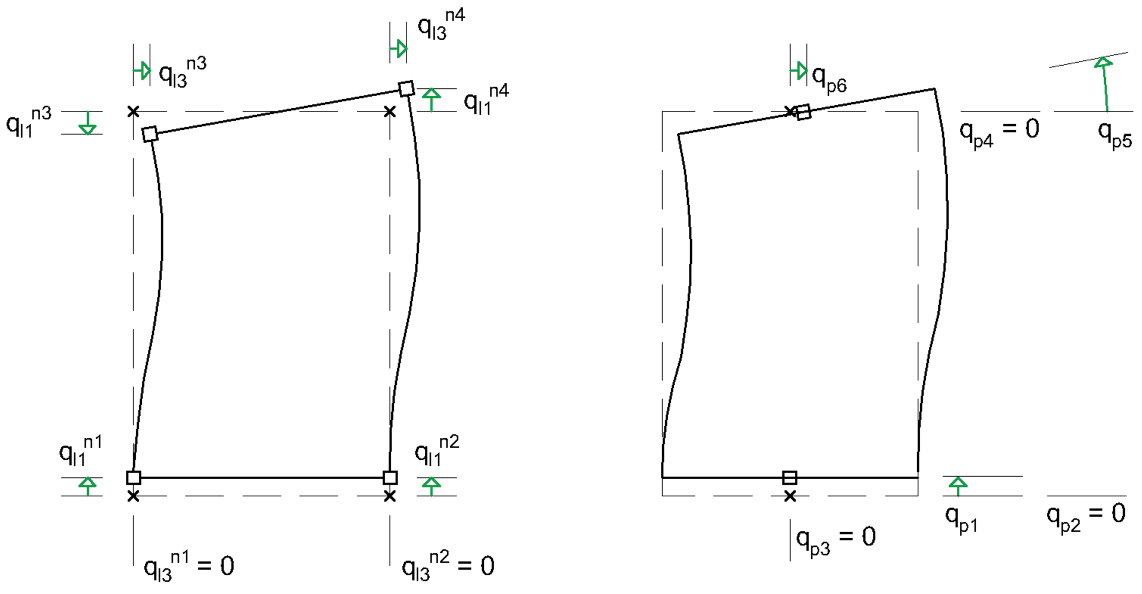


Figure 3.12 – Example of conversion from node displacements to wall displacements.

Writing the previous equations in matrix form:

$$\boldsymbol{q}_p = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix}}_{A_{ph}} \begin{bmatrix} q_{l3}^{n1} \\ q_{l3}^{n2} \\ q_{l3}^{n3} \\ q_{l3}^{n4} \end{bmatrix} + \underbrace{\begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ -1/b & 1/b & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & -1/b & 1/b \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{A_{pv}} \begin{bmatrix} q_{l1}^{n1} \\ q_{l1}^{n2} \\ q_{l1}^{n3} \\ q_{l1}^{n4} \end{bmatrix}$$

$$\boldsymbol{q}_p = \boldsymbol{A}_{ph}\, \boldsymbol{q}_h + \boldsymbol{A}_{pv}\, \boldsymbol{q}_v \tag{3.31}$$

With $\boldsymbol{q}_h$ corresponding to the horizontal displacements (which happen in the direction of $\overrightarrow{l_3}$), and $\boldsymbol{q}_v$ being the displacements in the vertical direction (along $\overrightarrow{l_1}$). In matrix form, this gives:

$$\boldsymbol{q}_h = \begin{bmatrix} q_{l3}^{n1} \\ q_{l3}^{n2} \\ q_{l3}^{n3} \\ q_{l3}^{n4} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{A}_{hl}} \boldsymbol{q}_l^w$$

$$\boldsymbol{q}_h = \boldsymbol{A}_{hl}\, \boldsymbol{q}_l^w \tag{3.32}$$

$$\boldsymbol{q}_v = \begin{bmatrix} q_{l1}^{n1} \\ q_{l1}^{n2} \\ q_{l1}^{n3} \\ q_{l1}^{n4} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}}_{\boldsymbol{A}_{vl}} \boldsymbol{q}_l^w$$

$$\boldsymbol{q}_v = \boldsymbol{A}_{vl}\, \boldsymbol{q}_l^w \tag{3.33}$$

Joining equations (3.32) and (3.33) into equation (3.31), one gets the relation between $\boldsymbol{q}_p$ and $\boldsymbol{q}_l^w$:

$$\boldsymbol{q}_p = \left(\boldsymbol{A}_{ph}\, \boldsymbol{A}_{hl}\right) \boldsymbol{q}_l^w + \left(\boldsymbol{A}_{pv}\, \boldsymbol{A}_{vl}\right) \boldsymbol{q}_l^w = \left(\boldsymbol{A}_{ph}\, \boldsymbol{A}_{hl} + \boldsymbol{A}_{pv}\, \boldsymbol{A}_{vl}\right) \boldsymbol{q}_l^w \tag{3.34}$$

Finally, using (3.27):

$$\boldsymbol{q}_p = \left(\boldsymbol{A}_{ph}\, \boldsymbol{A}_{hl} + \boldsymbol{A}_{pv}\, \boldsymbol{A}_{vl}\right) \boldsymbol{\Lambda}\, \boldsymbol{q}_e^w$$

Which results in a simpler expression following the definition of a transformation matrix $\boldsymbol{A}_w$:

$$\boldsymbol{A}_w = \left(\boldsymbol{A}_{ph}\, \boldsymbol{A}_{hl} + \boldsymbol{A}_{pv}\, \boldsymbol{A}_{vl}\right) \boldsymbol{\Lambda} \tag{3.35}$$

$$\boldsymbol{q}_p = \boldsymbol{A}_w\, \boldsymbol{q}_e^w \tag{3.36}$$

Going back to (3.29), having the stiffness matrix of the wall (beam stiffness matrix), one can write

$$\boldsymbol{K}_p\, \boldsymbol{q}_p = \boldsymbol{f}_p$$

Making use of equation (3.36):

$$\boldsymbol{K}_p\, (\boldsymbol{A}_w\, \boldsymbol{q}_e^w) = \boldsymbol{f}_p$$

Applying the Principle of Virtual Work:

$$\left(\boldsymbol{K}_p\, \boldsymbol{A}_w\, \boldsymbol{q}_e^w\right) \cdot \delta \boldsymbol{q}_p = \boldsymbol{f}_p \cdot \delta \boldsymbol{q}_p, \qquad \forall \delta \boldsymbol{q}_p$$

$$\left(\boldsymbol{K}_p\,\boldsymbol{A}_w\,\boldsymbol{q}_e^w\right)\cdot\left(\boldsymbol{A}_w\,\delta\boldsymbol{q}_e^w\right)=\boldsymbol{f}_p\cdot\left(\boldsymbol{A}_w\,\delta\boldsymbol{q}_e^w\right),\qquad\forall\delta\boldsymbol{q}_e^w$$

Where $(\cdot)$ stands for the inner product of two vectors. This means

$$\boldsymbol{A}_w^T\left(\boldsymbol{K}_p\,\boldsymbol{A}_w\,\boldsymbol{q}_e^w\right)\cdot\delta\boldsymbol{q}_e^w=\boldsymbol{A}_w^T\,\boldsymbol{f}_p\cdot\delta\boldsymbol{q}_e^w,\qquad\forall\delta\boldsymbol{q}_e^w$$

Which implies

$$\left(\boldsymbol{A}_w^T\,\boldsymbol{K}_p\,\boldsymbol{A}_w\right)\boldsymbol{q}_e^w=\boldsymbol{A}_w^T\,\boldsymbol{f}_p \tag{3.37}$$

Giving finally the stiffness matrix of the wall and the force vector in the $(\overrightarrow{e_1},\overrightarrow{e_2},\overrightarrow{e_3})$ frame, $\boldsymbol{K}_e^w$ and $\boldsymbol{f}_e^w$, respectively:

$$\boldsymbol{K}_e^w=\boldsymbol{A}_w^T\,\boldsymbol{K}_p\,\boldsymbol{A}_w \tag{3.38}$$

$$\boldsymbol{f}_e^w=\boldsymbol{A}_w^T\,\boldsymbol{f}_p \tag{3.39}$$

From these, the stiffness matrix of the whole core is assembled in a process analogous to the ones used for the frame substructures (n_w is the number of walls):

$$\boldsymbol{K}_c=\mathop{\mathsf{A}}_{w=1}^{\boldsymbol{n_w}}\boldsymbol{K}_e^w \tag{3.40}$$

The force vector $\boldsymbol{f}_c$ is assembled in a similar way, from the force vector of each wall, $\boldsymbol{f}_e^w$.

3.3.2 CONNECTED WALLS

In order to simulate certain configurations of core walls, it is helpful to introduce a second type of walls which do not connect directly to nodes but to other walls. This the case, for example, of an I type section, where the "flange" walls would need to be separated in two at the point where they meet the "web" wall

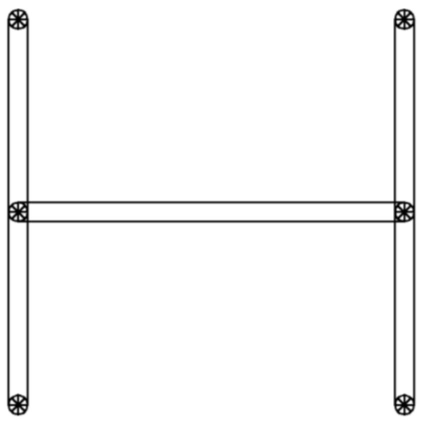


Figure 3.13 – Plan view of a regular I-section core.

in order to accommodate the nodes from the latter. This way, there are six independent nodes per floor, as can be seen on Figure 3.13.

If all the vertical displacements are considered independent, one is lead to an unwanted behavior in these walls, whereby they become disconnected along their height. This underestimates the stiffness associated to the core, especially when the “flange” wall is subject to axial stress, which leads to bending moment in the flange walls creating a discontinuity along their height:

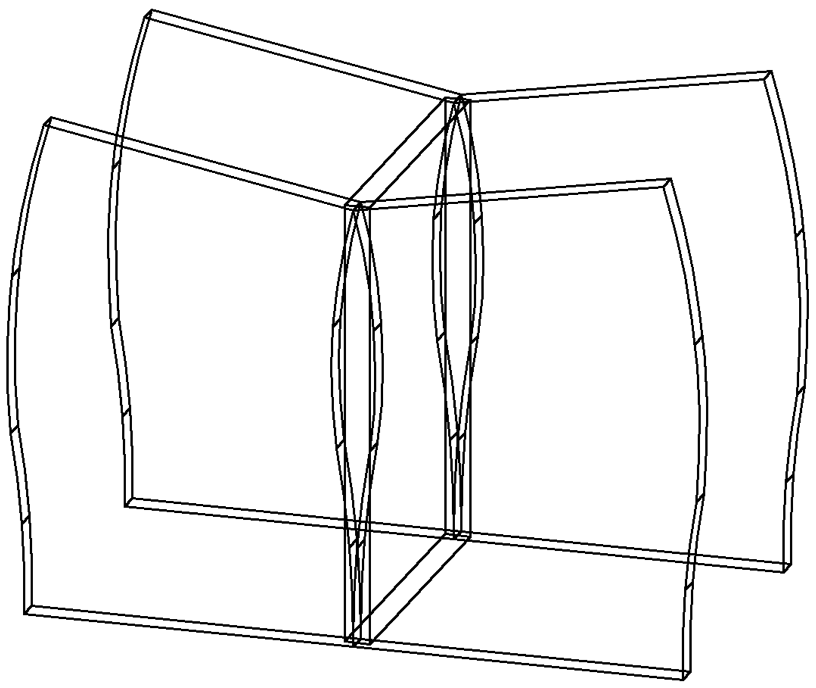


Figure 3.14 – I section core wall model subject to axial force.

However, it is possible to consider the stiffness of the “web” wall as acting directly on the nodes of the “flange” walls:

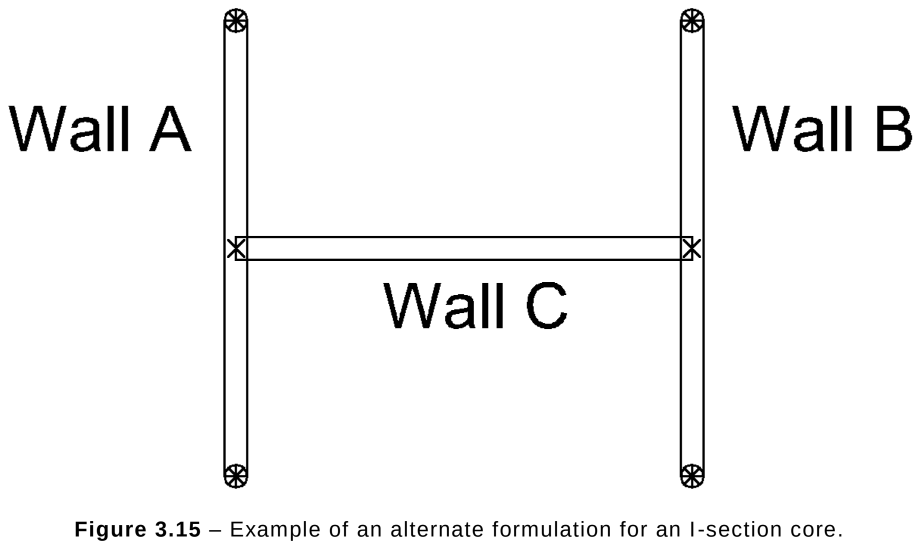


Figure 3.15 – Example of an alternate formulation for an I-section core.

This way, instead of six independent nodes per floor, there are only four, plus two new fictional nodes on the connections between the walls (as shown in Figure 3.15), whose displacements depend only on those of the four main modes.

Using the example above, the displacements of wall C, $\boldsymbol{q}_p$, can be written in the same manner as the other walls with an $\boldsymbol{A}_w$ transformation matrix (see equation (3.36)) but now considering nodes $n1$ through $n4$ as being four fictitious nodes, whose displacements depend on those from walls A and B, which gives, in the $(\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3})$ reference frame:

$$\begin{cases} \boldsymbol{q}_e^{n1} = \frac{1}{2}(\boldsymbol{q}_e^{n1} + \boldsymbol{q}_e^{n2})_A \\ \boldsymbol{q}_e^{n2} = \frac{1}{2}(\boldsymbol{q}_e^{n1} + \boldsymbol{q}_e^{n2})_B \\ \boldsymbol{q}_e^{n3} = \frac{1}{2}(\boldsymbol{q}_e^{n3} + \boldsymbol{q}_e^{n4})_A \\ \boldsymbol{q}_e^{n4} = \frac{1}{2}(\boldsymbol{q}_e^{n3} + \boldsymbol{q}_e^{n4})_B \end{cases} \tag{3.41}$$

Writing the system as a matrix:

$$\boldsymbol{q}_e^C = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_e^A \\ \boldsymbol{q}_e^B \end{bmatrix} = \boldsymbol{A}_{A,B}\, \boldsymbol{q}_e^{A,B} \tag{3.42}$$

Then, since a local frame can be defined for wall C, its $\boldsymbol{A}_w$ matrix can be computed. Hence, joining this last equation with equation (3.36) gives an expression for the $\boldsymbol{q}_p$ displacements in wall C:

$$\boldsymbol{q}_p = \left(\boldsymbol{A}_w\, \boldsymbol{A}_{A,B}\right) \boldsymbol{q}_e^{A,B} \tag{3.43}$$

Thus, using a reasoning similar to that of equation (3.38), the stiffness of wall B is given in the $(\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3})$ reference frame by:

$$\boldsymbol{K}_e^w = \left(\boldsymbol{A}_{A,B}^T\, \boldsymbol{A}_w^T\right) \boldsymbol{K}_p \left(\boldsymbol{A}_w\, \boldsymbol{A}_{A,B}\right) \tag{3.44}$$

Once again, each instance of $\boldsymbol{K}_e^w$ is assembled into $\boldsymbol{K}_c$, the total stiffness matrix of the core.

One of the advantages of this process is that it can also be generalized to any case where a wall meets another, regardless of the position and angle. It suffices that one ratio is given for each connection – that of the distance from node 1 of the connected wall to the connection, divided by the length of that same connected wall. This ratio, β, gives a weight for each of the sides of a connected wall and is used to define the matrix $\boldsymbol{A}_{A,B}$ As an example:

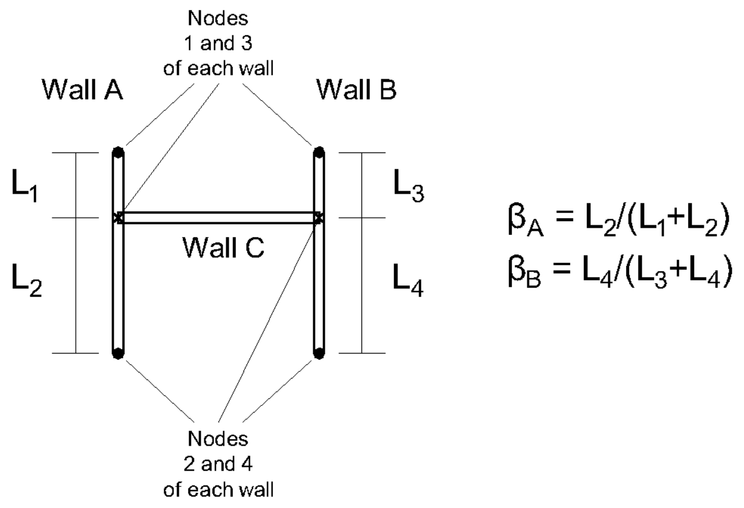


Figure 3.16 – Example of a generalization of the I-section core.

If β_A and β_B are the values of the previously mentioned ratio for the two connected walls A and B, then matrix $\boldsymbol{A}_{A,B}$ becomes:

$$\boldsymbol{A}_{A,B} = \begin{bmatrix} \beta_A & 1-\beta_A & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_A & 1-\beta_A & 0 & 0 \\ 0 & 0 & \beta_B & 1-\beta_B & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \beta_B & 1-\beta_B \end{bmatrix}$$

It follows that the I-shape is merely a particular case where $\beta_A = \beta_B = {}^1/_2$ and the connecting wall is perpendicular to both of the connected walls. Other common shapes can be modelled using this approach, such as a U-shaped section which could be modelled as two walls connected by a third with $\beta_A = \beta_B = 0$ (obviously, this particular case can also be studied as a regular wall).

This shows that the connected walls model gives a lot more flexibility regarding the possibilities for the modelling of walls.

The advantages that come from adding this type of wall will be analyzed and discussed in the chapter pertaining to the analysis of results.

3.3.3 STATIC CONDENSATION OF CORE WALLS

As rotations are not considered for this kind of sub-structure, the only “Slave” degrees of freedom are horizontal translations, as can be seen on Figure 3.17 (in red):

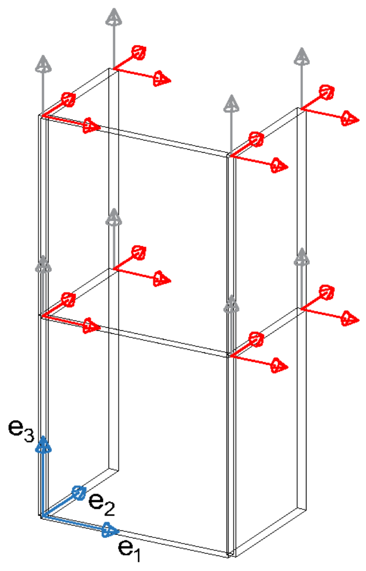


Figure 3.17 – “Slave” and “Local” displacements in a core wall.

Partitioning the matrix $\boldsymbol{K}_c$ according to equation (2.3) and performing static condensation on it in order to retrieve the system in terms of the horizontal displacements, gives once again:

$$\boldsymbol{K}_{ss}^{*}\,\boldsymbol{q}_s = \boldsymbol{f}_s^{*}$$

Which comes only in terms of $\boldsymbol{q}_s$. Once again, any pair of slave displacements on a given node can be expressed directly in terms of the global displacements $\boldsymbol{q}_g$ as:

$$\boldsymbol{q}_s^i = \begin{bmatrix} q_{e1}^i \\ q_{e2}^i \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha & d_1^{\perp} \\ -\sin\alpha & \cos\alpha & d_2^{\perp} \end{bmatrix} \begin{bmatrix} q_{x,j} \\ q_{y,j} \\ q_{\theta,j} \end{bmatrix}$$

As in the 3D frame case, $d^{\perp}$ is a signed distance to the global frame of reference given by one of the two following expressions:

$$\begin{cases} d_1^{\perp} = (\overrightarrow{g_3} \times \vec{x}) \cdot \overrightarrow{e_1} \\ d_2^{\perp} = (\overrightarrow{g_3} \times \vec{x}) \cdot \overrightarrow{e_2} \end{cases}$$

Once more, $\vec{x}$ is the position vector of the i^{th} node with respect to the global reference frame (see Figure 3.9).

The $\boldsymbol{A}_{sg}$ matrix is assembled just as in the previous cases, which then through equations (2.12) and (2.13) gives the contributions of the core wall to the stiffness and forces acting on the whole structure, respectively (in terms of the 3 degrees of freedom per story).

4 ANALYSIS

The analysis of the behavior of the model is divided in two parts: Static and Dynamic analysis.

The former evaluates the stresses and displacements in everyday use, while the latter considers specifically the effects of an earthquake. These two analyses can then be combined in order to give an overview of the viability of the structure.

4.1 Static Analysis

4.1.1 DEFINITION OF THE STATIC LOADS

For the static analysis, the combination used for loading is the frequent load, since the assumption is made that the most conditioning factor is the response due to dynamic loading. All the loads are considered as coming from the weight of the slabs. These weights can be summed per floor and divided by each corresponding surface area, giving the values of the loads in every story. Then, the loads can be divided among all sub-structures through the use of influence areas. A short summary is given for the procedure of each of the three types:

- Each 2D frame is assigned a width, so that the area of influence of each beam is simply its length multiplied by the width of the frame;
- On 3D frames, the area of influence of a beam is given by its length times half the sum of the distances to both of its neighboring beams;
- For core walls, the areas of influence of each node are defined, and the loads do not act directing on the walls but are instead acting on each node, as if they were external forces.

In the case of 2D and 3D frames, the area of influence of each beam is then multiplied by the load acting on the corresponding story, and as such, is accounted for in the assembly of the fixed end forces (see Sections 3.1.2 and 3.2.1)

On the other hand, for core walls, the area of influence of each node is multiplied by the corresponding load as well, but as it is now a force acting on a node, it is included as an external force.

4.1.2 COMPUTATION OF DISPLACEMENTS AND STRESSES

Once the full system of equations in terms of the 3 degrees of freedom per floor is assembled, the static analysis is performed by solving the system given by and obtaining the displacements on all degrees of freedom.

$$\boldsymbol{K}_{gg}\,\boldsymbol{q}_g = \boldsymbol{F}_g$$

From the solution $\boldsymbol{q}_g$, it is then possible to obtain the displacements on all nodes of every substructure by using its corresponding condensed system and $\boldsymbol{A}_{sg}$ matrix. Going back to equations (2.11) and (2.5), $\boldsymbol{q}_s$ and $\boldsymbol{q}_l$ are obtained through

$$\begin{cases} \boldsymbol{q}_s = \boldsymbol{A}_{sg}\, \boldsymbol{q}_g \\ \boldsymbol{q}_l = \boldsymbol{K}_{ll}^{-1}\, \boldsymbol{F}_l - \boldsymbol{K}_{ll}^{-1}\, \boldsymbol{K}_{ls}\, \boldsymbol{q}_s \end{cases}$$

These are all the nodal displacements in the sub-structure, and by knowing the end nodes of each element, it is possible to retrieve its nodal displacements, and therefore its stresses. This is done by multiplying the stiffness matrix of the element by its nodal displacements vector as seen with equations (3.7), (3.21) and (3.29):

$$\boldsymbol{f}_l^b = \boldsymbol{K}_l^b\, \boldsymbol{q}_l^b$$

$$\boldsymbol{f}_p = \boldsymbol{K}_p\, \boldsymbol{q}_p$$

This gives the vector of forces acting on the element's nodes, written as $\boldsymbol{f}_l^b$ for elements of 2D or 3D frames, and as $\boldsymbol{f}_p$ for wall elements. From here, knowing the loads acting on the element, the force diagrams can be traced out, giving the response of the structure under the static loading case, which is to be considered in combination with the dynamic case as well.

4.2 Dynamic Analysis

The dynamic analysis was carried by considering a standard model response spectrum analysis [4].

4.2.1 MULTI-DEGREE OF FREEDOM OSCILLATION

The dynamic analysis of the structure is performed by solving the system of differential equations for a multi-degree of freedom oscillator subject to a variable displacement:

$$\boldsymbol{M}\, \ddot{\boldsymbol{q}}(t) + \boldsymbol{C}\, \dot{\boldsymbol{q}}(t) + \boldsymbol{K}\, \boldsymbol{q}(t) = -\boldsymbol{M}\, \ddot{\boldsymbol{q}}_s(t) \qquad (4.1)$$

Where $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{C}$ is the damping matrix, and $\boldsymbol{K}$ is the stiffness matrix of each degree of freedom; $\boldsymbol{q}$ is the vector of relative displacements between the structure and the soil, and $\ddot{\boldsymbol{q}}_s$ is the acceleration at the ground level.

The stiffness matrix $\boldsymbol{K}$ is the exact same matrix used in the static analysis ($\boldsymbol{K}_{gg}$), and the mass matrix $\boldsymbol{M}$ is calculated assuming that all the mass of the building is lumped at the level of the floors. The matrix entry $\boldsymbol{M}_{ij}$ is the force (or moment) felt in the i^{th} degree of freedom due to a unit acceleration (or unit angular acceleration) along the j^{th} degree of freedom. This results in the following mass matrix for each slab:

$$\boldsymbol{M}_{slab} = \begin{bmatrix} M_x & 0 & -S_y \\ 0 & M_y & S_x \\ -S_y & S_x & I_\theta \end{bmatrix}$$

Where $M_x = M_y$ is the mass of the slab, S_x and S_y are the static moments around global axes x and y; and I_θ is the moment of inertia of the slab around the origin of the reference frame.

The full mass matrix of the structure is then obtained by going slab by slab and assembling the respective mass matrices in the corresponding 3x3 sets of and columns along the diagonal:

$$\boldsymbol{M} = \begin{bmatrix} \left[\sum_{1st\,floor} \boldsymbol{M}_{slab}\right] & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \left[\sum_{nth\,floor} \boldsymbol{M}_{slab}\right] \end{bmatrix}$$

In order to obtain the modes of vibration of the structure, one can perform the calculations using equation (4.1) with $\boldsymbol{C} = 0$ since the typical value of damping on a reinforced concrete building is around 5%, and therefore the error introduced by ignoring it is indeed very small [4].

The solution for the system of differential equations

$$\boldsymbol{M}\,\ddot{\boldsymbol{q}}(t) + \boldsymbol{K}\,\boldsymbol{q}(t) = 0 \tag{4.2}$$

Is assumed to be of the form

$$\boldsymbol{q}(t) = \boldsymbol{q}\,cos(pt - \varphi) \tag{4.3}$$

Where $\boldsymbol{q}$ is a vector that represents the deformed configuration of the degrees of freedom, p is the angular frequency, t is time and φ is the phase shift (the starting position of the structure in the oscillatory movement).

This implies

$$\ddot{\boldsymbol{q}}(t) = -p^2\,\boldsymbol{q}\cos(pt - \varphi) = -p^2\,\boldsymbol{q}(t) \tag{4.4}$$

Taking this last equation and introducing it into (4.2), one gets

$$\boldsymbol{K}\,\boldsymbol{q}(t) - \boldsymbol{M}\,p^2\boldsymbol{q}(t) = 0$$

$$\boldsymbol{K}\,\boldsymbol{q}\cos(pt-\varphi) = p^2\boldsymbol{M}\,\boldsymbol{q}\cos(pt-\varphi)$$

$$\boldsymbol{K}\,\boldsymbol{q} = p^2\,\boldsymbol{M}\,\boldsymbol{q} \tag{4.5}$$

Where $\boldsymbol{q} \neq 0$.

The problem of finding the values of p^2 and corresponding vectors $\boldsymbol{q}$ which satisfy the previous equation is known as the generalized eigenvalue problem and it has been extensively studied, and MATLAB has a predefined function (the command "eig") which allows to find its solution. Since both matrices are symmetric and real-valued, there will be as many distinct solutions as the dimension of the matrices, that is, as the number of degrees of freedom. These $({p_i}^2, \boldsymbol{q}_i)$ pairs correspond to the square of the frequency and shape vector of the i^{th} mode respectively, and it will now be shown that any oscillatory behavior of the structure can be considered as a superposition of these modes.

Two important things should be noted about the eigenvectors:

Firstly, they are orthogonal with respect to both matrices $\boldsymbol{M}$ and $\boldsymbol{K}$. This means that if $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ are two linearly independent vectors satisfying the equation, then:

$$\begin{cases}\boldsymbol{K}\,\boldsymbol{q}_i = {p_i}^2\,\boldsymbol{M}\,\boldsymbol{q}_i \\ \boldsymbol{K}\,\boldsymbol{q}_j = {p_j}^2\,\boldsymbol{M}\,\boldsymbol{q}_j\end{cases} \Leftrightarrow \begin{cases}\boldsymbol{K}\,\boldsymbol{q}_i = {p_i}^2\,\boldsymbol{M}\,\boldsymbol{q}_i \\ \boldsymbol{q}_j^T\boldsymbol{K} = {p_j}^2\,\boldsymbol{q}_j^T\boldsymbol{M}\end{cases} \Leftrightarrow \begin{cases}\boldsymbol{q}_j^T\,\boldsymbol{K}\,\boldsymbol{q}_i = {p_i}^2(\boldsymbol{q}_j^T\,\boldsymbol{M}\,\boldsymbol{q}_i) \\ \boldsymbol{q}_j^T\,\boldsymbol{K}\,\boldsymbol{q}_i = {p_j}^2(\boldsymbol{q}_j^T\,\boldsymbol{M}\,\boldsymbol{q}_i)\end{cases}$$

$$\begin{cases}{p_i}^2(\boldsymbol{q}_j^T\,\boldsymbol{M}\,\boldsymbol{q}_i) = {p_j}^2(\boldsymbol{q}_j^T\,\boldsymbol{M}\,\boldsymbol{q}_i) \\ \dfrac{1}{{p_i}^2}(\boldsymbol{q}_j^T\,\boldsymbol{K}\,\boldsymbol{q}_i) = \dfrac{1}{{p_j}^2}(\boldsymbol{q}_j^T\,\boldsymbol{K}\,\boldsymbol{q}_i)\end{cases} \Leftrightarrow \begin{cases}({p_i}^2 - {p_j}^2)(\boldsymbol{q}_j^T\,\boldsymbol{M}\,\boldsymbol{q}_i) = 0 \\ \left(\dfrac{1}{{p_i}^2} - \dfrac{1}{{p_j}^2}\right)(\boldsymbol{q}_j^T\,\boldsymbol{K}\,\boldsymbol{q}_i) = 0\end{cases}$$

Which finally, for $p_i \neq p_j$ yields the aforementioned orthogonality conditions:

$$\begin{cases}\boldsymbol{q}_j^T\,\boldsymbol{M}\,\boldsymbol{q}_i = 0 \\ \boldsymbol{q}_j^T\,\boldsymbol{K}\,\boldsymbol{q}_i = 0\end{cases} \tag{4.6}$$

The second thing to notice is that the i^{th} mode shape vector $\boldsymbol{q}_i$ can be multiplied by a scalar, and still remain a solution to the previous equation. Hence, the modal shapes are determined in terms of the relation between all the degrees of freedom for any given mode, but not in terms of their absolute values. This allows for each of the $\boldsymbol{q}_i$ to be normalized in the following manner:

$$\boldsymbol{\phi}_i = \frac{\boldsymbol{q}_i}{\sqrt{\boldsymbol{q}_i^T\,\boldsymbol{M}\,\boldsymbol{q}_i}} \tag{4.7}$$

This normalization has very nice properties:

$$\boldsymbol{\phi}_i^T \boldsymbol{M} \boldsymbol{\phi}_i = \frac{\boldsymbol{q}_i^T \boldsymbol{M} \boldsymbol{q}_i}{\left(\sqrt{\boldsymbol{q}_i^T \boldsymbol{M} \boldsymbol{q}_i}\right)^2} = 1 \tag{4.8}$$

$$\boldsymbol{\phi}_i^T \boldsymbol{K} \boldsymbol{\phi}_i = \frac{\boldsymbol{q}_i^T \boldsymbol{K} \boldsymbol{q}_i}{\left(\sqrt{\boldsymbol{q}_i^T \boldsymbol{M} \boldsymbol{q}_i}\right)^2} = {p_i}^2 \frac{\boldsymbol{q}_i^T \boldsymbol{M} \boldsymbol{q}_i}{\left(\sqrt{\boldsymbol{q}_i^T \boldsymbol{M} \boldsymbol{q}_i}\right)^2} = {p_i}^2 \tag{4.9}$$

And finally, assuming that the damping matrix $\boldsymbol{C}$ is a linear combination of the mass and stiffness matrices, it can be diagonalized using the same basis [4]:

$$\boldsymbol{\phi}_i^T \boldsymbol{C} \boldsymbol{\phi}_i = \frac{\boldsymbol{q}_i^T \boldsymbol{C} \boldsymbol{q}_i}{\left(\sqrt{\boldsymbol{q}_i^T \boldsymbol{M} \boldsymbol{q}_i}\right)^2} = 2\xi_i p_i \frac{\boldsymbol{q}_i^T \boldsymbol{M} \boldsymbol{q}_i}{\left(\sqrt{\boldsymbol{q}_i^T \boldsymbol{M} \boldsymbol{q}_i}\right)^2} = 2\xi_i p_i \tag{4.10}$$

where ξ_i is the viscous damping ratio, given by $\xi_i = \frac{c_i}{c_{cr,i}}$, with c_i and $c_{cr,i}$ being respectively the modal damping and the critical modal damping, the latter being equal to $2p_i$.

It becomes useful now to consider the matrix $\boldsymbol{\Phi}$, which contains the normalized eigenvectors as its columns:

$$\boldsymbol{\Phi} = \begin{bmatrix} | & | & \cdots & | \\ \boldsymbol{\phi}_1 & \boldsymbol{\phi}_2 & \cdots & \boldsymbol{\phi}_n \\ | & | & \cdots & | \end{bmatrix}$$

This matrix can be used to diagonalize $\boldsymbol{M}$, $\boldsymbol{K}$, and $\boldsymbol{C}$ with:

$$\boldsymbol{\Phi}^T \boldsymbol{M} \boldsymbol{\Phi} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{4.11}$$

$$\boldsymbol{\Phi}^T \boldsymbol{K} \boldsymbol{\Phi} = \begin{bmatrix} {p_1}^2 & 0 & \cdots & 0 \\ 0 & {p_2}^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & {p_m}^2 \end{bmatrix} \tag{4.12}$$

$$\boldsymbol{\Phi}^T \boldsymbol{C} \boldsymbol{\Phi} = \begin{bmatrix} 2\xi_1 p_1 & 0 & \cdots & 0 \\ 0 & 2\xi_2 p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2\xi_m p_m \end{bmatrix} \tag{4.13}$$

where m is the number of modes, equal to the number of global degrees of freedom.This is indeed a useful result, since one can now treat $\boldsymbol{\Phi}$ as a basis of the space of configurations of the displacements of the structure, having as new coordinates the amplitudes of each mode (in terms of the normalized basis), and using the following equations in order to switch between displacement and modal coordinates:

$$\boldsymbol{q} = \boldsymbol{\Phi}\, \boldsymbol{q}_G \quad ; \quad \boldsymbol{q}_G = \boldsymbol{\Phi}^{-1}\boldsymbol{q} \tag{4.14}$$

This in turn means that the former system of differential equations seen in equation (4.1) can be thought of as a set of disjoint 1-degree of freedom systems, one for each mode, all acting simultaneously in the structure:

$$\boldsymbol{M}\,\ddot{\boldsymbol{q}} + \boldsymbol{C}\,\dot{\boldsymbol{q}} + \boldsymbol{K}\,\boldsymbol{q} = -\boldsymbol{M}\,\ddot{\boldsymbol{q}}_s$$

$$\boldsymbol{\Phi}^T\boldsymbol{M}(\boldsymbol{\Phi}\,\boldsymbol{\Phi}^{-1})\ddot{\boldsymbol{q}} + \boldsymbol{\Phi}^T\boldsymbol{C}(\boldsymbol{\Phi}\,\boldsymbol{\Phi}^{-1})\dot{\boldsymbol{q}} + \boldsymbol{\Phi}^T\boldsymbol{K}(\boldsymbol{\Phi}\,\boldsymbol{\Phi}^{-1})\boldsymbol{q} = -\boldsymbol{\Phi}^T\boldsymbol{M}\,\ddot{\boldsymbol{q}}_s$$

$$\begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} \ddot{\boldsymbol{q}}_G + \begin{bmatrix} 2\xi_1 p_1 & & \\ & \ddots & \\ & & 2\xi_m p_m \end{bmatrix} \dot{\boldsymbol{q}}_G + \begin{bmatrix} {p_1}^2 & & \\ & \ddots & \\ & & {p_m}^2 \end{bmatrix} \boldsymbol{q}_G = -\boldsymbol{\Phi}^T\boldsymbol{M}\,\ddot{\boldsymbol{q}}_s \tag{4.15}$$

Focusing now the right-hand side of the equation, the value of the ground acceleration $\ddot{q}_s$ can be split into its three components $\ddot{q}_{sx}$, $\ddot{q}_{sy}$ and $\ddot{q}_{sz}$. Then, the vector $\ddot{\boldsymbol{q}}_s$ becomes:

$$\ddot{\boldsymbol{q}}_s = \mathbb{1}_x\,\ddot{q}_{sx} + \mathbb{1}_y\,\ddot{q}_{sy} + \mathbb{1}_z\,\ddot{q}_{sz} = [\mathbb{1}_x \quad \mathbb{1}_y \quad \mathbb{1}_z] \begin{bmatrix} \ddot{q}_{sx} \\ \ddot{q}_{sy} \\ \ddot{q}_{sz} \end{bmatrix} \tag{4.16}$$

Where $\mathbb{1}_x$, $\mathbb{1}_y$, $\mathbb{1}_z$ are column vectors with ones in the degrees of freedom aligned with the x, y and z directions, respectively. In this case, since the degrees of freedom considered are only horizontal, only the x and y parts of the equation need to be taken into account.

The right-hand side of (4.15) becomes:

$$-\boldsymbol{\Phi}^T\boldsymbol{M}\,\ddot{\boldsymbol{q}}_s = -\boldsymbol{\Phi}^T\boldsymbol{M}\left(\mathbb{1}_x\,\ddot{q}_{sx} + \mathbb{1}_y\,\ddot{q}_{sy}\right) = -\boldsymbol{\Phi}^T\boldsymbol{M}\,\mathbb{1}_x\,\ddot{q}_{sx} - \boldsymbol{\Phi}^T\boldsymbol{M}\,\mathbb{1}_y\,\ddot{q}_{sy} \tag{4.17}$$

The modal participation factors for mode i are then defined as:

$$\begin{cases} P_{ix} = \boldsymbol{\phi}_i^T\boldsymbol{M}\,\mathbb{1}_x \\ \\ P_{iy} = \boldsymbol{\phi}_i^T\boldsymbol{M}\,\mathbb{1}_y \end{cases} \tag{4.18}$$

The equation (4.15) can then finally be simplified to an equation describing a single degree of freedom per mode:

$$\ddot{q}_{Gi} + (2\xi_i p_i)\dot{q}_{Gi} + ({p_i}^2)q_{Gi} = -P_{ix}\,\ddot{q}_{sx} - P_{iy}\ddot{q}_{sy} \tag{4.19}$$

Solving all equations separately yields the response in terms of the amplitude of each mode. This can then be combined using equation (4.14) to get the response in terms of the actual displacements along the degrees of freedom.

4.2.2 RESPONSE SPECTRA ANALYSIS

In order to model the effects of the ground acceleration, EC-8 recommends the use of an elastic response spectrum, which gives the expected maximum acceleration felt in the structure due to a seismic event. From the values for the frequency and the damping, the response spectrum provides the spectral acceleration, S_d:

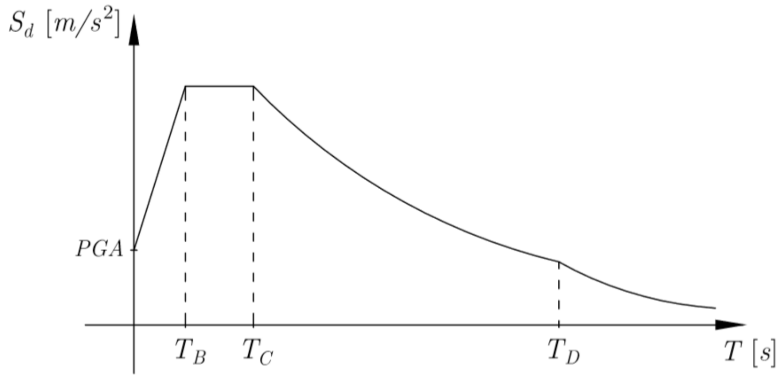


Figure 4.1 – Response spectrum from Eurocode 8.

The periods for each mode are easily computed, with

$$T_i = \frac{2\pi}{p_i} \tag{4.20}$$

EC8 specifies two locations and five ground types which define the response spectrum:

Type 1 (distant earthquake):

Table 4.1 – Soil coefficient and periods for a type 1 earthquake

Ground Type	S	$T_B(s)$	$T_C(s)$	$T_D(s)$
A	1.0	0.1	0.6	2.0
B	1.35	0.1	0.6	2.0
C	1.6	0.1	0.6	2.0
D	2.0	0.1	0.8	2.0
E	1.8	0.1	0.6	2.0

Type 2 (near):

Table 4.2 – Soil coefficient and periods for a type 2 earthquake

Ground Type	S	$T_B(s)$	$T_C(s)$	$T_D(s)$
A	1.0	0.1	0.25	2.0
B	1.35	0.1	0.25	2.0
C	1.6	0.1	0.25	2.0
D	2.0	0.1	0.3	2.0
E	1.8	0.1	0.25	2.0

From these values, the spectral acceleration, S_d, can be found by resorting to the following equations:

$$\begin{cases} 0 < T < T_B : S_d(T) = a_g \cdot S \cdot \left[\frac{2}{3} + \frac{T}{T_B} \cdot \left(\frac{2.5}{q} - \frac{2}{3}\right)\right] \\ T_B < T < T_C : S_d(T) = a_g \cdot S \cdot \frac{2.5}{q} \\ T_C < T < T_D : S_d(T) = a_g \cdot S \cdot \frac{2.5}{q} \cdot \left[\frac{T_C}{T}\right] \\ T_D < T : S_d(T) = a_g \cdot S \cdot \frac{2.5}{q} \cdot \left[\frac{T_C T_D}{T^2}\right] \end{cases} \tag{4.21}$$

It is to be noted that even though S_d depends on both T_i and ξ_i, the expressions are only explicitly written in terms of T_i because the effects of ξ_i are being accounted for by the behavior factor, q.

From here, assuming that the earthquake is acting in the x direction, and taking the response spectrum obtained:

$$\begin{cases} \ddot{q}_{Gi_{max}} = P_{ix} S_d \\ q_{Gi_{max}} = \frac{P_{ix} S_d}{{p_i}^2} \end{cases} \tag{4.22}$$

Knowing that $q_{Gi_{max}}$ is the maximum amplitude of the i^{th} mode in the normalized coordinates, the actual displacements are retrieved by multiplying this amplitude by its corresponding shape vector:

$$\boldsymbol{q}_{max} = \boldsymbol{\phi_i} \frac{P_{ix} S_d}{{p_i}^2} \tag{4.23}$$

Then, the corresponding forces and stresses in each degree of freedom and each element can be computed in a similar way to the one which was used for the static analysis. For any sub-structure, $\boldsymbol{q}_s$ and $\boldsymbol{q}_l$ are obtained through equations (2.11) and (2.5), knowing the corresponding matrix $\boldsymbol{A}_{sg}$:

$$\begin{cases} \boldsymbol{q}_s = \boldsymbol{A}_{sg}\, \boldsymbol{q}_g \\ \boldsymbol{q}_l = \boldsymbol{K}_{ll}^{-1}\, \boldsymbol{F}_l - \boldsymbol{K}_{ll}^{-1}\, \boldsymbol{K}_{ls}\, \boldsymbol{q}_s \end{cases}$$

Since for the seismic response the vector $\boldsymbol{F}_l$ is zero, a simplified expression for the “Local” degrees of freedom can be written as:

$$\boldsymbol{q}_l = -\,\boldsymbol{K}_{ll}^{-1}\,\boldsymbol{K}_{ls}\,\boldsymbol{q}_s \tag{4.24}$$

Now, having the values of all the displacements along degrees of freedom of the sub-structure (because vectors $\boldsymbol{q}_l$ and $\boldsymbol{q}_s$ make up a partition of the vector of all degrees of freedom of the sub-structure), the forces acting on each element can be retrieved by multiplying the stiffness matrix of the element by its displacements vector, as seen in equations (3.7), (3.21) and (3.29):

$$\boldsymbol{f}_l^b = \boldsymbol{K}_l^b\,\boldsymbol{q}_l^b$$

$$\boldsymbol{f}_p = \boldsymbol{K}_p\,\boldsymbol{q}_p$$

From here, the results are combined to get an estimate of the stress on every element. It is important to stress that the combination should always be the last calculation to perform, meaning that one should first calculate all forces due to displacements and only then combine them, instead of combining the displacements and calculating the resulting forces afterwards.

The analysis for an earthquake acting along the y direction is completely analogous to the one just presented. The final values of stresses and displacements are arrived at through the combination of the effects of an earthquake in both directions.

4.2.3 COMBINATION OF RESULTS FROM THE SPECTRAL ANALYSIS

Combination of modes

The combination of the stress-resultants from each mode is performed with the Complete Quadratic Combination (CQC). This method is more reliable than other common alternatives, since it takes into account the correlation between modes, and thus gives better results for cases where the modes are not completely independent from each other, or when torsional effects are significant.

For n modes, the CQC combination of a set of forces from n different modes acting on any given point of the structure is done in the following manner:

$$F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{m} F_i F_j \mu_{ij}} \tag{4.25}$$

Where F_i and F_j are the forces felt on the i^{th} and j^{th} modes and μ_{ij} is the correlation between modes i and j, given by:

$$\mu_{ij} = \frac{8\xi^2(1+r)r^{\frac{3}{2}}}{(1-r^2)^2 + 4\xi^2 r(1+r)^2} \quad ; \quad r = \frac{p_i}{p_j} \tag{4.26}$$

The resulting force, F, is thus the combined response from all modes due to an earthquake in the direction considered.

Combination of effects in both directions

With the results from the effects due to the seismic components along x and y, the final values of stresses and displacements are computed using the square root of sum of squares method (SRSS):

$$F = \sqrt{{F_x}^2 + {F_y}^2} \tag{4.27}$$

This choice comes from the assumption that the effects due to an earthquake in both directions are independent from each other and with the same intensity.

It should be pointed out that the values that arise from this expression are always non-negative, hence their sign unknown, which makes it necessary to consider both possibilities while studying the interaction of different stresses on any particular element.

4.3 Analysis of stresses

The analysis of the structure is done by considering the effects of both the static loads and the effects due to an earthquake. The superposition of these two cases can be calculated for any element by considering the stresses due to the static case, and then checking all possible combinations with different signs for the dynamic case. Since the aim is to detect which elements are overstressed, in practice, one needs only to estimate the maximum stresses, disregarding their signs. Thus, the bending moment is simply given by

$$M = |M_S| + |M_D|$$

where subscripts S and D refer to the static and dynamic stresses, respectively. For the columns, the true sign of the static loads should be taken into account and combined with both possibilities for the sign of the axial stresses resulting from the dynamic analysis, since it is not possible to know in advance which one is the most critical.

This results in two possibilities for the axial stress, N_1 and N_2, given by:

$$N_1 = N_S + |N_D|$$

$$N_2 = N_S - |N_D|$$

These are then used to evaluate the level of stress, as well as to give an estimate of how much steel reinforcement each beam or column needs.

4.3.1 ANALYSIS OF BEAMS

Since beam elements are not expected to have high values of axial stress, the effects of interaction between axial stress and bending moment are not taken into account. This makes the process easier, since all that is needed is comparing values of bending with the resistant capacity of the element.

The analysis is done resorting to three dimensionless parameters: the mechanical reinforcement ratio ω, the relative moment μ, and the relative axial stress ν. These are defined by the following expressions:

$$\omega = \frac{A_s}{bh}\frac{f_{yd}}{f_{cd}}\ ; \qquad \mu = \frac{M}{bd^2 f_{cd}}\ ; \qquad \nu = \frac{N}{bdf_{cd}} \tag{4.28}$$

This provides a way to find an estimate for the area of steel reinforcement needed, given by:

$$A_s = \omega bh \frac{f_{cd}}{f_{yd}} \tag{4.29}$$

In the case of bending moment acting along one of the principal inertia axes, with no axial stress, the mechanical reinforcement ratio is given by [5]:

$$\omega = \frac{1 - \sqrt{1 - 2.42\mu}}{1.21} \tag{4.30}$$

Now, using equation (4.29), the value for the area of steel reinforcement can be found.

There is no need to differentiate beams belonging to 3D frames from those belonging to 2D frames, since both are expected to have very low values of moment in the horizontal plane.

4.3.2 ANALYSIS OF COLUMNS

Since column elements have both axial stress and bending moment, their interaction needs to be accounted for. This is done using an interaction diagram, which gives the recommended mechanical reinforcement ratio, ω, in terms of the relative moment μ and the relative axial stress ν. An example is given below (taken from [6]):

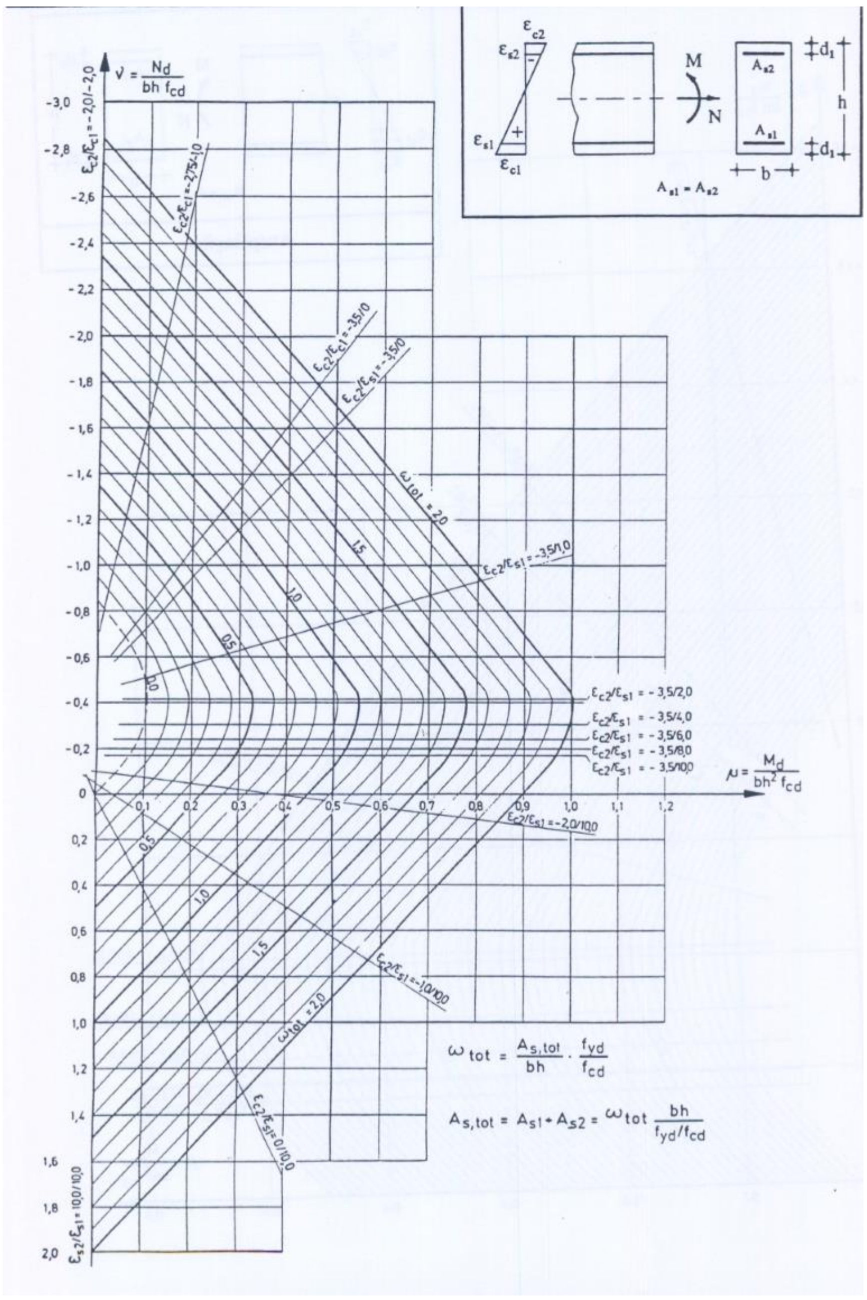


Figure 4.2 – Diagram for calculating the mechanical reinforcement ratio in an element.

From the value of ω obtained from the diagram and using again equation (4.29), it is possible to arrive at an estimate of how much reinforcement the element needs.

In the case of elements belonging to 3D frames, the moments in both directions have to be accounted for, and a simplification was made by assuming $\mu = \sqrt{\mu_2{}^2 + \mu_3{}^2}$. This simplification means that all directions in which the bending moments can occur are considered as being the same in terms of the interactions between axial stress and moment. In reality, this is only expected to happen in circular cross-sections, but it still remains a good approximation when applied to rectangular cross-sections.

In order to check that the simulation is well done and that the structure is viable, a few indicators are set up:

- The value of ω should not be higher than 0.35;
- The value of μ should not be higher than 0.25;
- The value of ν should be very low on beams, below 0.01;
- The value of moment M_3 should be very low on beams.

The first two conditions, which were adapted from [5], show that the structural design is appropriate, while the last two should always be true. If any of these are not verified, there is some chance that either there is a problem with the simulation, or the modelled structure is not well designed. The program is designed to output a message in case this occurs.

4.3.3 DISCRETIZATION OF THE INTERACTION DIAGRAM

When doing seismic design, one of the most difficult problems is to come up with combination rules that properly account for the correlation of modal contributions. This has been addressed in [7], [8] and [9]. Here, an alternative approach is taken where instead of finding the response interaction diagram, one firstly discretizes the capacity interaction diagram into several linear functions to then be able to apply simple modal combinations.

The diagram mentioned in the previous section used for determining the mechanical reinforcement ratio from the values of μ and ν has to be discretized in the code, and this discretization can be defined by giving a series of planes defined by linear equations of the form

$$\omega = \vec{x} \cdot \vec{t} + C \tag{4.31}$$

Where the input vector $\vec{x}$ is a (μ, ν) pair, $\vec{t}$ is a 2D vector which defines both a direction in the $\mu - \nu$ plane and a slope; and C is a scalar constant. When several of these planes are defined, one computes the values of ω for all such equations and picks the maximum among them.

The specific discretization used in the code presented in Figure 4.2 is taken from [6]. It assumes the reinforcement is A400 steel and the value of d_1/h is around 0.05 (see box in top right corner of Figure 4.2). However, it is possible to define a different one by giving, for each plane, any set of three non-collinear points as well as their ω values:

If three points P, Q and R with coordinates (μ_P, ν_P), (μ_Q, ν_Q) and (μ_R, ν_R) are given, the scalar C and vector $\vec{t}$ that define equation (4.31) can be obtained by solving the following system of equations for t_1, t_2 and C:

$$\begin{cases} \vec{P} \cdot \vec{t} + C = \omega_P \\ \vec{Q} \cdot \vec{t} + C = \omega_Q \\ \vec{R} \cdot \vec{t} + C = \omega_R \end{cases} \Leftrightarrow \begin{cases} \mu_P t_\mu + \nu_P t_\nu + C = \omega_P \\ \mu_Q t_\mu + \nu_Q t_\nu + C = \omega_Q \\ \mu_R t_\mu + \nu_R t_\nu + C = \omega_R \end{cases} \qquad (4.32)$$

This approach is rather useful, since it allows for the discretization of the interaction diagram to any degree of approximation required, simply by adding more planes. An example is given below in order to illustrate the process:

Supposing the diagram from Figure 4.2 needs to be discretized in two planes as such:

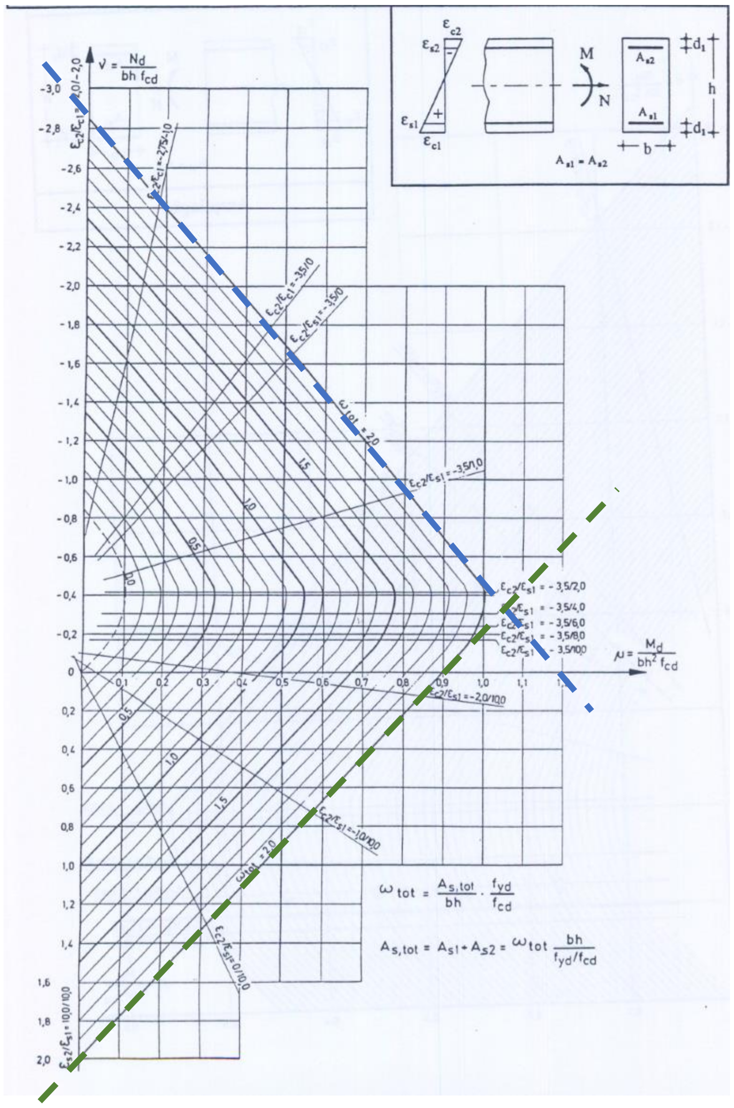


Figure 4.3 – Discretization of an interaction diagram using two planes.

Then, for the first plane (shown in blue in Figure 4.6):

$$\begin{cases} (\mu_P, \nu_P) = (1.20\,, 0.00) \\ (\mu_Q, \nu_Q) = (0.00\,, -2.85) \\ (\mu_R, \nu_R) = (0.40\,, -1.40) \end{cases} \quad ; \quad \begin{cases} \omega_P = 2 \\ \omega_Q = 2 \\ \omega_R = 1.5 \end{cases}$$

Solving the system in (4.32) and thus obtaining the vector $\vec{t}$ and scalar C that define the mechanical reinforcement ratio in plane 1:

$$\begin{cases} t_{\mu,1} = 2.375 \\ t_{\nu,1} = -1 \\ C_1 = -0.85 \end{cases}$$

Equation (4.31) now gives values which when plotted look like the following graph:

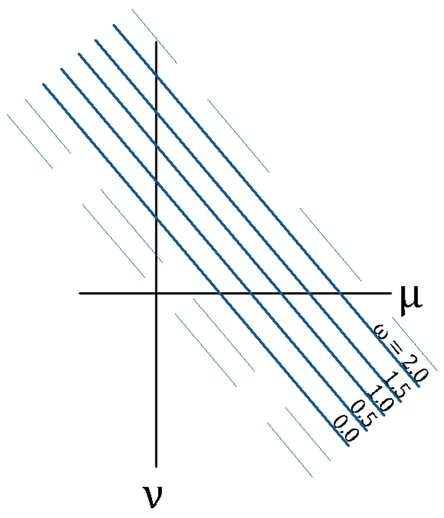


Figure 4.4 – Example of a plane defining a linear equation.

Similarly, for the second plane (shown in green in Figure 4.6):

$$\begin{cases} (\mu_P, \nu_P) = (0.00\,, 0.00) \\ (\mu_Q, \nu_Q) = (0.90\,, 0.00) \\ (\mu_R, \nu_R) = (0.00\,, 2.00) \end{cases} \quad ; \quad \begin{cases} \omega_P = 0 \\ \omega_Q = 2 \\ \omega_R = 2 \end{cases}$$

Again, when system (4.32) is solved, its results allow the for another equation of the form presented in (4.31) to be defined, this time giving the following values:

$$\begin{cases} t_{\mu,2} = 2.222 \\ t_{\nu,2} = 1 \\ C_2 = 0 \end{cases}$$

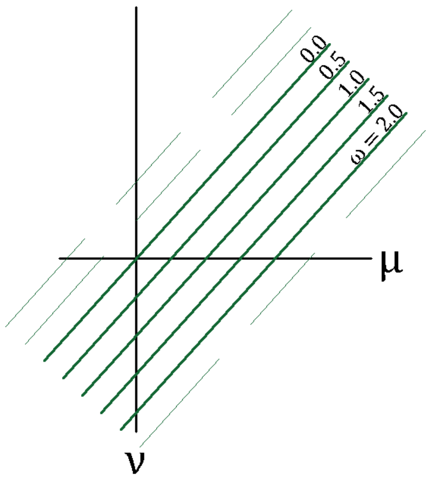


Figure 4.5 – A second plane discretization.

When combined, the graphs of both equations resemble the original one, albeit with a somewhat sizeable error in the area closer to the μ axis:

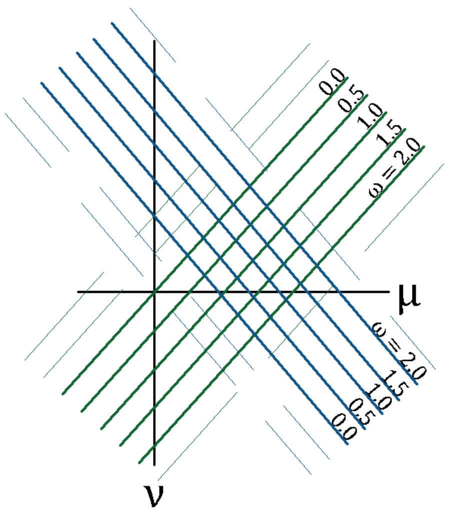


Figure 4.6 – Superposition of the two previously defined equations.

This error could be further diminished by the introduction of a third plane, approximating better the area in question.

From here, the value of ω for any combination of μ and ν can be found by computing the equation (4.31) for all the planes in the discretization and taking the maximum of all such values. As a final example, considering an element with the following relative stresses:

$$(\mu, \nu) = (0.15\ , -0.60)$$

Using the equation found for plane 1:

$$\omega_1 = \vec{x} \cdot \vec{t}_1 + C_1 = 0.15 \times 2.375 + (-0.60) \times (\ -1) - 0.85 = 0.1062$$

For plane 2:

$$\omega_2 = \vec{x} \cdot \vec{t}_2 + C_2 = 0.15 \times 2.222 + (-0.60) \times 1 + 0 = -0.2667$$

The value for the mechanical reinforcement ratio for this element is thus

$$\omega = \max(\omega_1, \omega_2) = 0.1062$$

5 IMPLEMENTATION

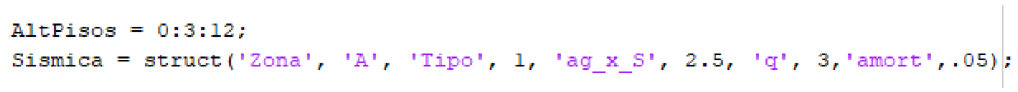
The language chosen for the implementation of this method into a computer program was MATLAB [10], mainly due to its efficiency in dealing with matrices and systems of equations. An object-oriented approach was taken, which means that new types of variables were defined, called “classes”. The instances of such classes are called objects, and any object of a certain class contains data, in the form of attributes (also called “Properties”); and code, in the form of procedures (“Methods”). Properties are fields specific to each class which store information about the object. Methods are routines that can be performed on objects of that class, namely reading and editing data from the property fields of said objects. All classes also have a special method which is called the “constructor”, which is the routine that creates an object of the class and outputs it.

This approach allows for several similar substructures, for example 2D frames, to be recognized and treated as objects of only one class, “2D Frame”, meaning that they share a list of common properties and can thus be analyzed using the same tools (the methods written for that class). The approach was chosen mainly due to the already well-defined types of components considered in the modelling of structures (walls, 2D and 3D frames) and the highly repetitive nature of the problem.

Other advantages of object-oriented programming are encapsulation, which means that classes can keep their objects’ properties private from the rest of the code - only allowing them to be edited through calling its specific procedures; and the modularity that comes with it, which opens the possibility for new classes to be defined and added to the program without interfering with the previously written code.

This division of data in classes is of course hidden from the user, who only needs to input one predefined MATLAB data file with the following information:

1. Heights of floors relative to the ground level;
2. Characteristics of the soil and dynamic behavior of the building:
    2.1. Seismic zone of the construction;
    2.2. Type of earthquake considered for the seismic analysis;
    2.3. Ground/Soil base acceleration;
    2.4. Behavior coefficient of the building (q);
    2.5. Damping factor of the structure;



```
AltPisos = 0:3:12;
Sismica = struct('Zona', 'A', 'Tipo', 1, 'ag_x_S', 2.5, 'q', 3,'amort',.05);
```

3. Characteristics of the slabs (per slab):
    3.1. Length in both x and y directions;
    3.2. Position of the center of mass of the slab;

3.3. The mass of the slab, including the equivalent mass of all loads applied on it;

```
Lajes(1) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 650000, 'h', 3);
Lajes(2) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 650000, 'h', 6);
Lajes(3) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 650000, 'h', 9);
Lajes(4) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 650000, 'h', 12);
```

4. Definition of the types of cross-section used in the building:
   4.1. Width and height of the rectangular cross section;
   4.2. Elastic modulus of the concrete;
   4.3. Maximum admissible stress of the concrete;

```
Sec(1) = struct('B', .45, 'H', .45, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6); %%%% Pilares
Sec(2) = struct('B', .3, 'H', .45, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6); %%%% Vigas
```

5. Characteristics of all 2D frames (per frame):
   5.1. Direction of the plane, given as the coordinates of a vector with the intended direction;
   5.2. Position of the 2D reference frame in the global reference frame of the model;
   5.3. Distances between columns;
   5.4. Floors spanned by the 2D frame;
   5.5. Cross-section types of columns and beams;
   5.6. Forces applied on nodes of the frame;

```
Portico(1) = struct('dir', [1, 0], 'origem', [-12.5, -13], 'vaos', [5, 5, 5, 5, 5],...
    'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 2.5, 'Fn', []);
```

6. Characteristics of all 3D frames:
   6.1. Position of the 3D reference frame in the global reference frame of the model;
   6.2. Distances between columns in both the x and y directions;
   6.3. Floors spanned by the frame;
   6.4. Cross-section types of beams and columns;
   6.5. Forces applied on nodes;

```
Portico3D(1) = struct('vaos_x', [5, 5, 5, 5, 5], 'vaos_y', [5, 5, 6, 5, 5],...
    'Pisos', 1:5, 'SecPilares', 1, 'SecVigas', 2, 'Pos_ref', [-12.5; -13; 0],'Fn',[]);
```

7. Characteristics of all wall cores:
   7.1. Coordinates of the nodes in the x-y plane;
   7.2. Walls connecting nodes (given as the numbers for the start and end nodes);
   7.3. Walls connecting to other walls;
   7.4. Thickness of the walls;
   7.5. Floors spanned by the core;
   7.6. Elastic Modulus of the concrete.

```
Nucleo(1) = struct('Nos', [-1, -2; 1, -2; -1, 2; 1, 2], 'Paredes', [1, 2; 3, 4],...
    'ParedesLigadas', [1,.5,2,.5], 't', 0.05, 'Pisos', 1:7, 'E', 30 * 10 ^ 9,...
    'fcd', 30 * 10 ^ 6, 'A_inf', [4.5, 5]);
```

8. Choice of plots to be presented:

```
Graficos = struct('ModosPlanta', 1:3, 'AnimPlanta', [], 'Anim3D', [],...
    'Def', 3, 'Cond', false, 'Esf', true);
```

The program then takes the data on this list and chooses the relevant information for the characterization of each class, passing it to the respective constructors, and receiving back the initialized objects.

Five different types of class were created:

- Cross-Section;
- 2D Frame;
- 3D Frame;
- Core wall;
- Structure;

Some common features can be found among the "2D Frame", "3D Frame" and "Wall Core" classes, especially the first two; for since they represent the three main types of substructure, similar procedures were defined for them, such as the computation of stresses from displacements, and the graphic visualization of the structure. These five classes will now be explained in detail.

5.1 Classes

5.1.1 CROSS-SECTION CLASS

The Cross-section class was created to store data relating to the geometry and physical properties of elements, and to perform calculations which involve the specific details of the cross sections, namely computing the relative stresses and the area of steel reinforcement needed (see Section 4.3). It works under the assumption that sections are rectangular and made of concrete, although the code could later be improved upon in order to deal with different types of material.

Its constructor receives input data of type (4) and uses it to compute and store the necessary property values of the object, such as area, moments of inertia, elastic modulus and maximum admissible stress. It is outputted and given as input to the 2D and 3D frame class constructors and saved in their properties. Whenever a calculation involving a specific beam or column needs to be performed, the corresponding cross-section object is called and the relevant properties are read, such as when assembling defining the stiffness matrix of each element; or a method is performed, for example when the safety verification is done given the forces acting on any particular beam.

5.1.2 2D FRAME CLASS

This is the class dealing with frames contained in a single plane. Its constructor receives data input of type (1), (4) and (5), and proceeds to create an object of the class "2D Frame". An array with information about the nodes is created and saved as one of the properties of the object. This array contains the numbering of the nodes, their coordinates, the applied forces and the displacements which are fixed for each node. From here, the number of degrees of freedom of the substructure can be counted.

After this, a similar variable defining columns and beams is created and saved to the properties of the object as well. It identifies for each element, its start and end nodes, the cross-section type and the loads acting on it.

The rest of the information needed to perform the analysis is then computed, as described in Section 4.1: The stiffness matrices of each beam and columns are assembled into the 2D Frame stiffness matrix, $\boldsymbol{K}_f$, which is saved as a property of the object; and the fixed end actions and nodal force vectors are assembled and the former is subtracted from the latter to get the force vector of the frame, $\boldsymbol{f}_f$. Static condensation is performed by checking which of the numbered degrees of freedom lie along the horizontal direction and extracting the sub-matrices $\boldsymbol{K}_{ss}$, $\boldsymbol{K}_{ll}$ and $\boldsymbol{K}_{sl}$ from the stiffness matrix of the frame, and vectors $\boldsymbol{f}_s$ and $\boldsymbol{f}_l$ from the force vector $\boldsymbol{f}_f$. From these, following the method presented in Section 2.2.2, matrix $\boldsymbol{K}_{ss}^*$ and $\boldsymbol{f}_s^*$ are computed and stored in the properties, as well as the transformation matrix $\boldsymbol{A}_{sg}$ which is used to arrive to the contributions to the 3 degree per story model, $\boldsymbol{K}_{gg}$ and $\boldsymbol{f}_g$. The initialization of the frame object is finished, and the constructor outputs the object of type “3D Frame” which is to be given as input to the “Structure” class constructor.

Other methods that were defined to act on objects of this type include a procedure that given the displacements in the structure, outputs back the values of stress in every element; and one that performs the modal and directional combinations of stresses and displacements.

5.1.3 3D FRAME CLASS

Objects of this class represent the three-dimensional frames in the structure. The class shares much of its structure with the previous one, due to the similarities of solving 2D and 3D frames.

Its constructor receives data input of types (1), (4) and (6), and creates and numbers the nodes and beams, as well as calculating and saving their defining properties, in much the same fashion as its 2D counterpart. The only main differences arise in the size of preallocated memory for matrices and vectors, since for every node there are now six degrees of freedom to consider; and in the assembly of the transformation matrix $\boldsymbol{A}_{sg}$ used to write “Slave” coordinates in terms of “Global” coordinates (see Section 2.2.3).

5.1.4 CORE WALL CLASS

This is the class used to represent the other type of common substructure in a building: Core and shear walls. These objects also share some features with the frame classes, mainly the methods.

The constructor for objects of this class starts by receiving the input data from (1) and (7), and just as in the two previous cases, it creates a variable that stores the coordinates of each node. From there, and also using the input given, a new variable containing the data defining the walls is stored in the properties. This variable is then used to save the stiffness matrices of the walls once they are computed, as well as their transformation matrix, $\boldsymbol{A}_w$ (see Section 3.3.1). The stiffness matrix of the core, $\boldsymbol{K}_c$, is assembled and after static condensation is finished, its $\boldsymbol{K}_{gg}$ and $\boldsymbol{A}_{sg}$ matrices are saved.

5.1.5 STRUCTURE CLASS

This is the class that contains all of the information about the structure being modelled. It aggregates all instances of other classes used and makes operations on them, calling them in order.

Its constructor receives as input the data from (1), (2) and (3), as well as an array of elements of each sub-structure class, already initialized. All the contributions from the substructures, $\boldsymbol{K}_{gg}$ and $\boldsymbol{f}_g$ are summed up to give the whole static system of equations of the model. The static analysis is then performed by resorting to MATLAB's "matrix division" which is an efficient way to solve systems of equations, aiming to minimize computing time. The values of $\boldsymbol{q}_g$, which are the displacements due to the static loading, are saved in the properties of the object.

Since each substructure class has a method defined to calculate the forces due to displacements, these are called to compute and store in the object the values of stresses in all elements.

Afterwards, the mass matrix, $\boldsymbol{M}_{gg}$ is assembled from the input data defining the slabs. These are considered to be rectangular, and the contribution of each slab to the mass matrix is assembled as described in Section 4.2.1. The mass and stiffness matrices are then used to find the modes of the structure through the MATLAB command "eig", which solves the general eigenvalue problem discussed in Section 4.2.1. The shape vectors are normalized and saved, as well as the corresponding frequencies. Having the modes, equations (4.18) are used to calculate the modal participation factors.

The response spectrum is chosen using the input data contained in (2). Then, from the frequency of each mode, its corresponding period is computed, and the spectrum is used to find the value for the spectral acceleration, S_d, for each mode. The displacements due to each mode are saved.

Once again, each class's method is called to find out the stresses due to the displacements, but this time it has to be done considering each mode separately, as well both horizontal directions possible for the earthquake. After that, the modes are combined, and then the directions, again with methods developed for each class.

Finally, the verification procedures are run, which involve going through each element in every substructure and checking the amount of stress and the area of reinforcement needed to see if the resulting values are acceptable in terms of the design, as described in Section 4.3.

The following class diagram illustrates the relationships between classes:

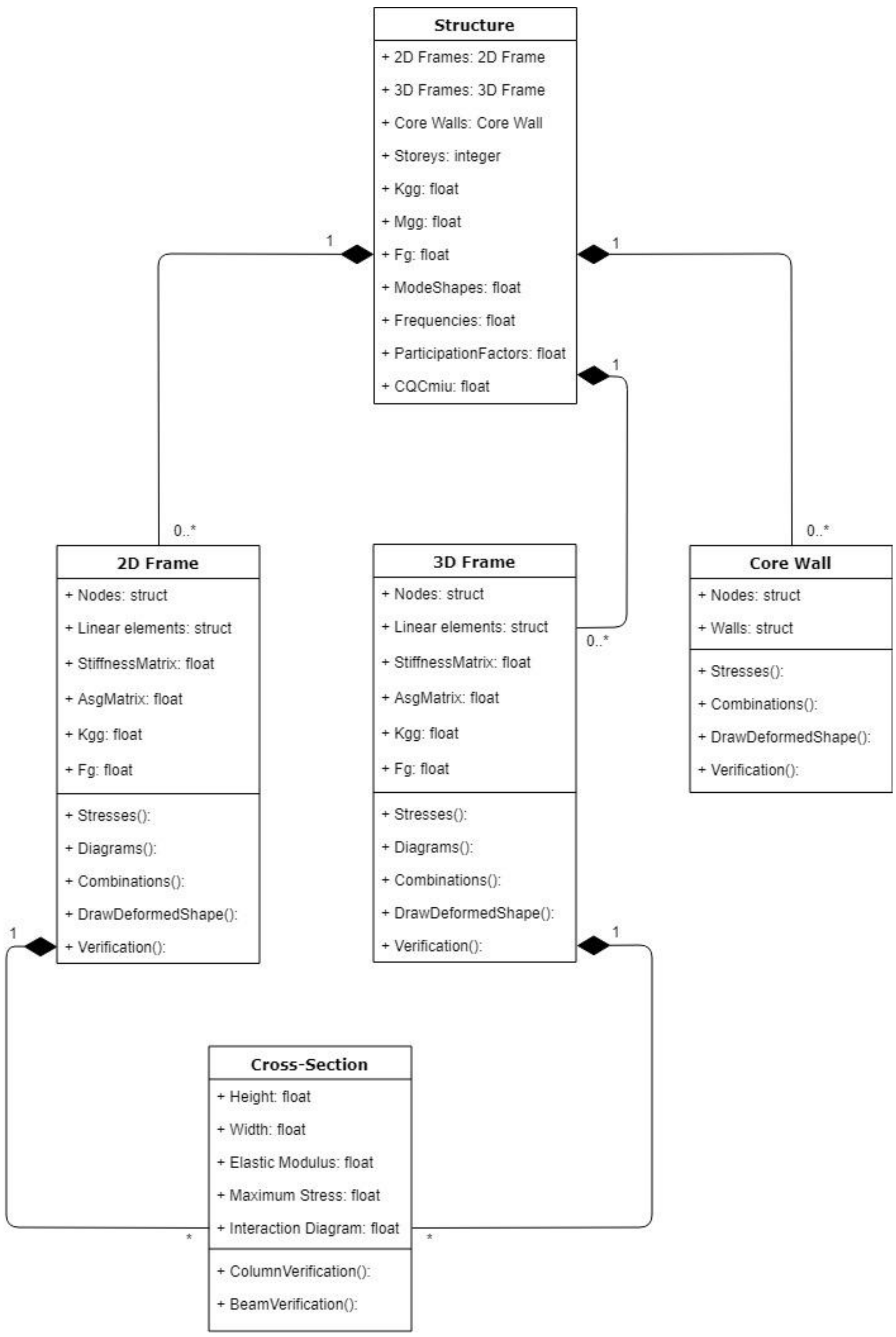


Figure 5.1 – Class diagram showcasing the different classes.

5.2 Output

Once the analysis is complete, several plots are presented to give an overview of the structure:

- A plan view gives the position of the slabs, in as many modes as required;

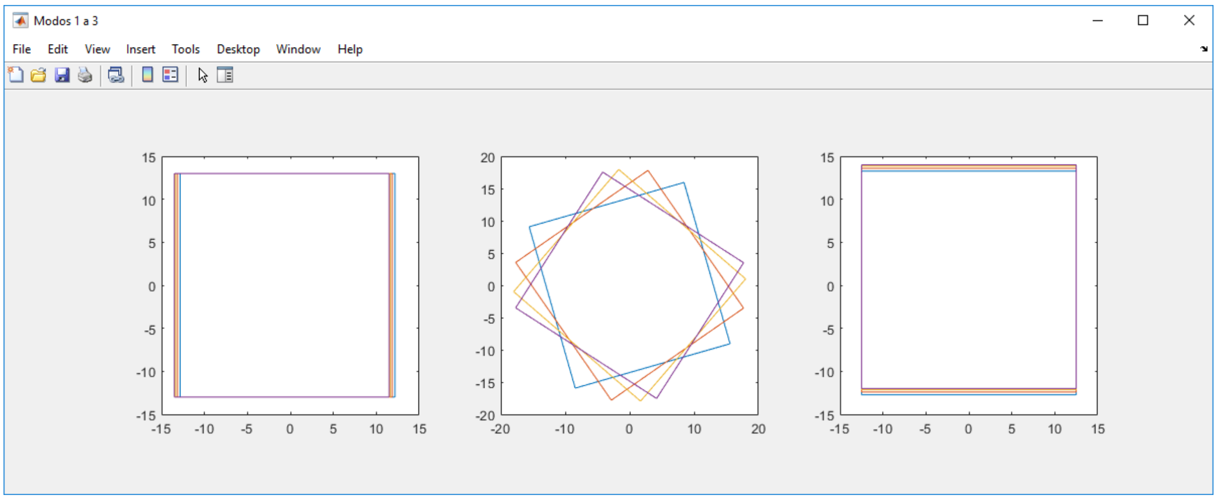


Figure 5.2 – Plot of a plan view showing the modal shapes of a model.

- An animation of the plan view of the slabs for a specific mode;

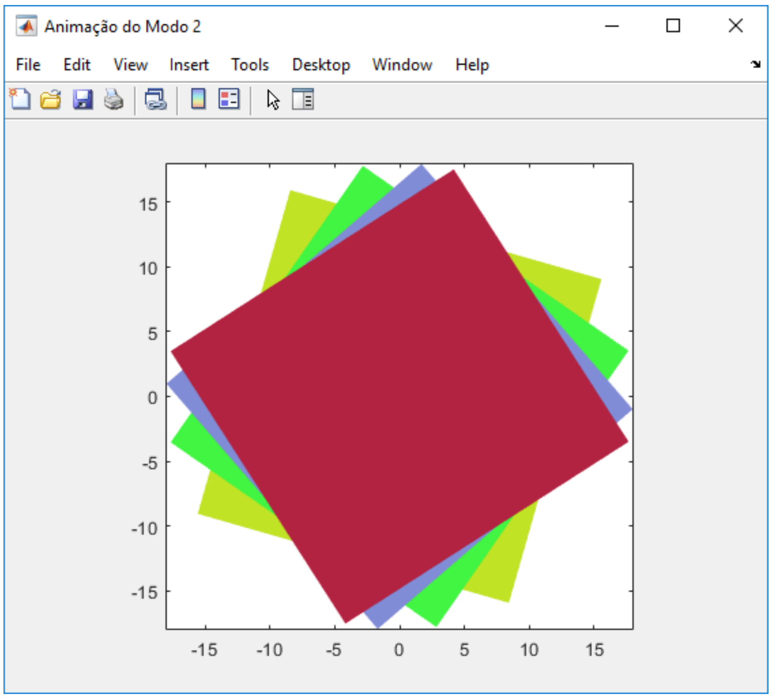


Figure 5.3 – Animation of the shape of a mode.

- A 3D plot of the structure with the deformed configuration of the structure for any specific mode;

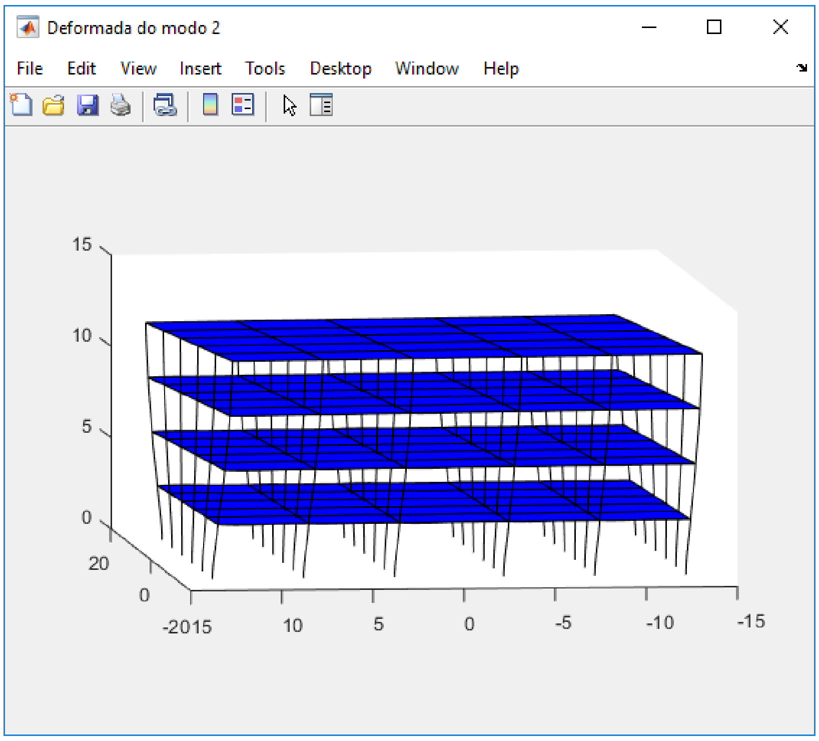


Figure 5.4 – 3D plot of the deformed configuration of a mode.

- A 3D plot showing the structure with color indicating the amount of stress level;

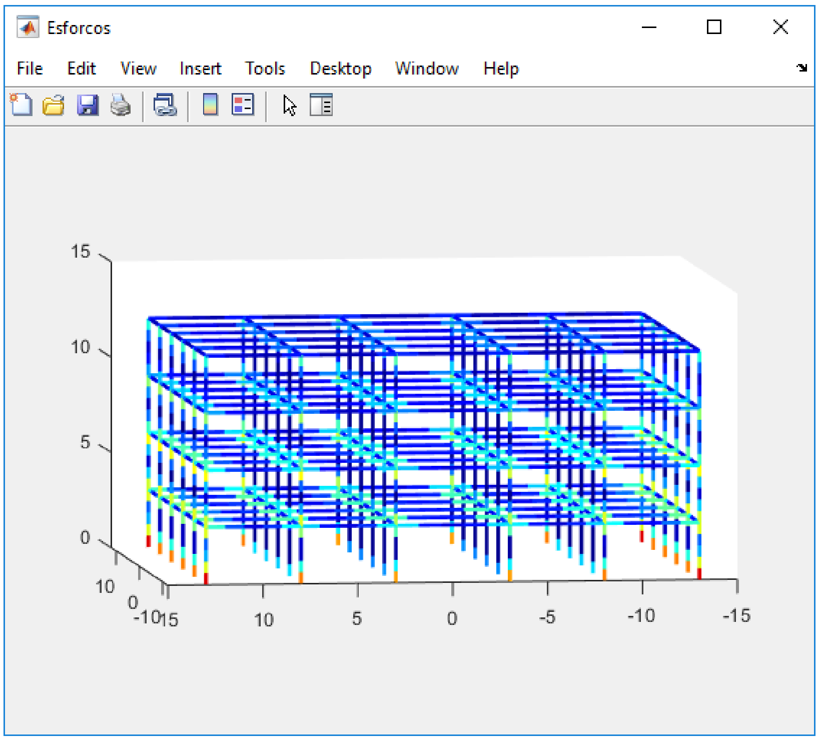


Figure 5.5 – Colored 3D plot showing stress level indicators using a color scale.

- A 3D plot of the structure with over-stressed elements colored in red and the rest in green, using the indicators defined in Section 4.3;

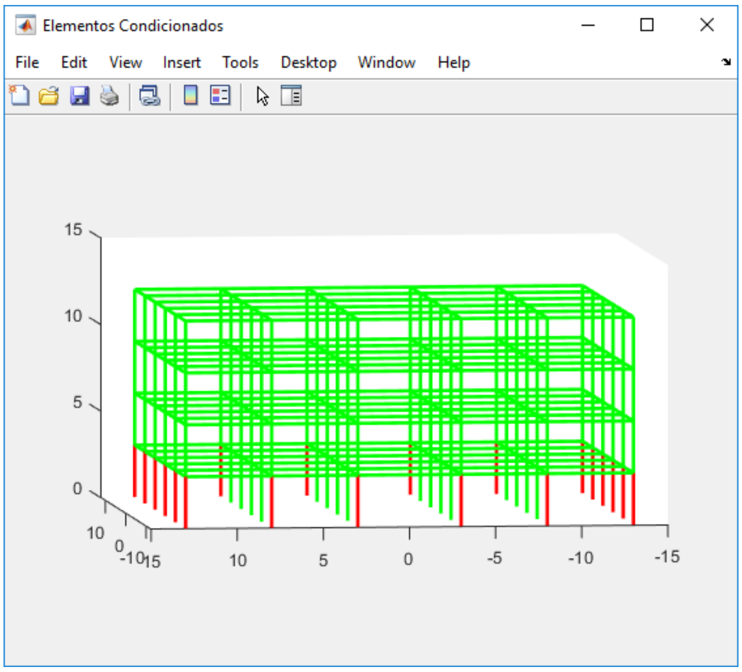


Figure 5.6 – 3D plot showing in red the elements with stresses exceeding admissible levels.

- A 3D animation of the slabs of the structure in any specific mode;

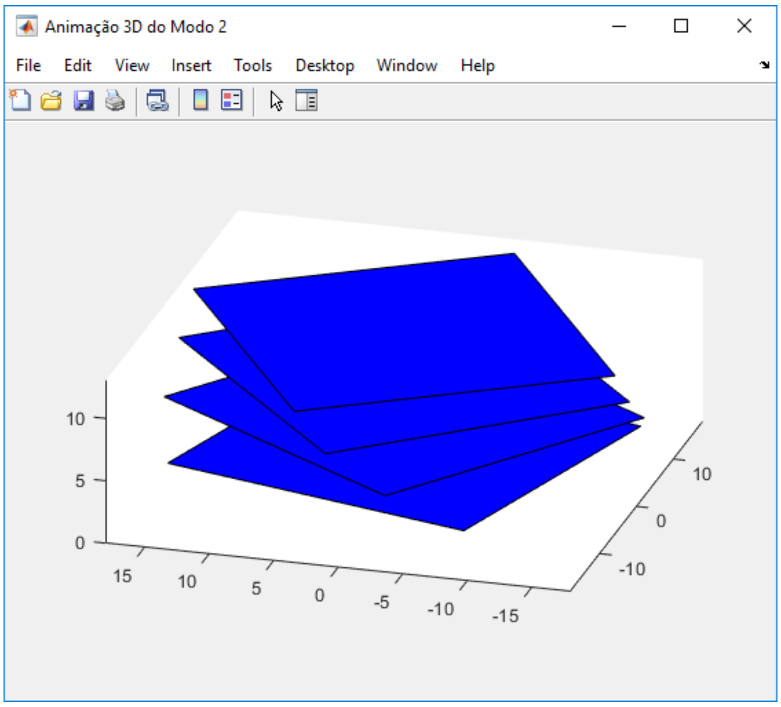


Figure 5.7 – 3D plot showing an animation of the displacements of the slabs.

All these plots are optional and any combination of them can be asked for in the input data.

6 TESTS AND ANALYSIS OF RESULTS

Once the program was implemented, several models were tested in order to evaluate the correctness of the results, ranging all types of substructures used:

- Comparison between the same regular structure modelled with 2D and 3D frames;
- Comparison between the behavior of I-section cores with connected walls and with regular walls;
- Modelling of an asymmetric and irregular building in height, made from 2D frames and a core wall;
- Analysis of an alternative method for computing stresses and the mechanical reinforcement ratio.

The results of these tests are discussed in detail in this chapter, comparing them with expected results, and evaluating the efficiency of the analysis. In all models, the height between floors is 3 meters, and the loads are 10 kN/m^2. The seismic zone is considered to be of type A and the modelling is done for an earthquake of type 1 (distant). The behavior factor is taken as 3, and the damping ratio as 5%. For the analysis of the cross-sections, the concrete has $f_{cd} = 30\ \mathrm{MPa}$, and the steel reinforcement $f_{yd} = 348\ \mathrm{MPa}$.

6.1 3D Frames vs. Several 2D Frames

A simple structure consisting of a regular 3D Frame was compared with a similar one built from 2D Frames, with the intent to quantify the differences between the modelling of both. The building is detailed below (all dimensions presented are in meters):

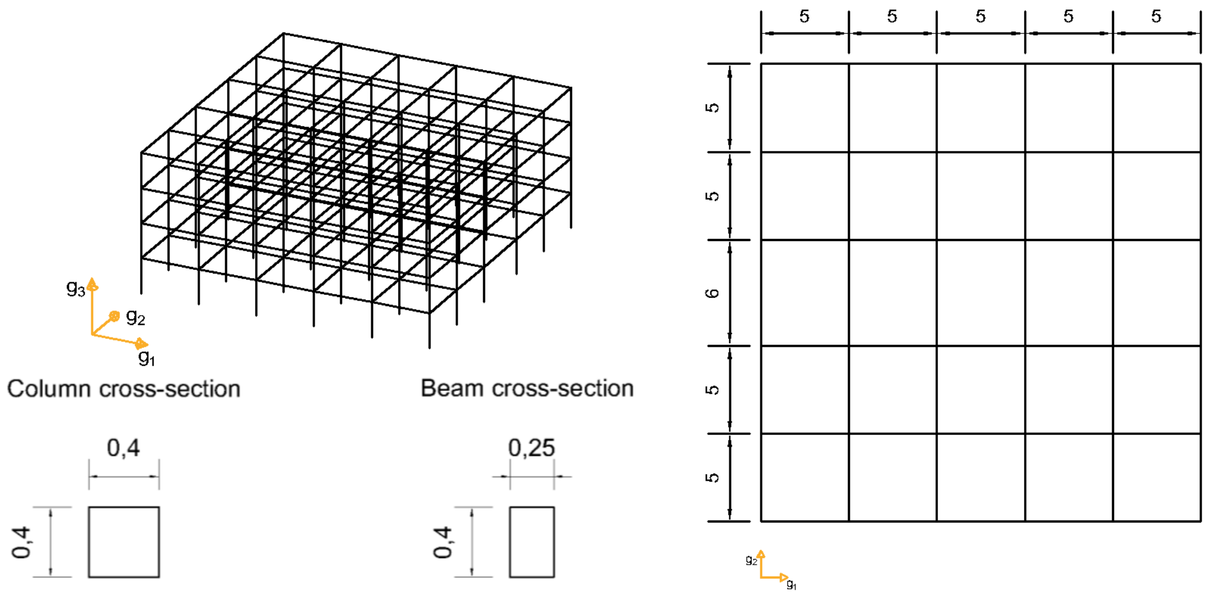


Figure 6.1 – Details of the model.

In terms of the seismic response, as expected, the only divergences between the two appeared on the frequencies of the torsional modes, with the 3D Frames behaving slightly better than their 2D counterparts due to the torsional rigidity of 3D elements being taken into account.

Table 6.1 – Comparison between the seismic response for both models

	Effective Masses		Frequencies [Hz]		
Mode	Mx	My	2D	3D	% error
1	0,00%	82,71%	1,8757	1,8757	0,00%
2	82,81%	0,00%	1,8946	1,8946	0,00%
3	0,00%	0,00%	2,2443	2,2701	1,13%
4	0,00%	11,40%	6,1918	6,1918	0,00%
5	11,34%	0,00%	6,2380	6,2380	0,00%
6	0,00%	0,00%	7,3985	7,4604	0,83%
7	0,00%	4,47%	11,7486	11,7486	0,00%
8	4,44%	0,00%	11,7917	11,7917	0,00%
9	0,00%	0,00%	14,0103	14,0733	0,45%
10	0,00%	1,42%	17,3301	17,3301	0,00%
11	1,40%	0,00%	17,3452	17,3452	0,00%
12	0,00%	0,00%	20,6359	20,6858	0,24%

However, the main difference appeared in the static analysis. If 2D frames are considered as crossing each other, then when columns of a certain 3D frame belong to more than one 2D frame, (which is the case for all the columns in this particular model), then the same column would have to belong to more than one of 2D frames, when being modelled in that way.

The result is that some columns are double-counted, which means that in the 2D frames case there are two columns considered for every column actually in the model, but with only half the load acting on each. Consequently, the stress due to the static load in columns is underestimated when a frame is modelled in such a way.

An even worse shortcoming was that since 2D frame elements are modelled as only having moment in the plane of the frame, each of the copies of a column will only account for the moment in one direction, thus completely losing the possibility of evaluating the interaction between moments in both directions and axial stresses.

This prompted a search for a solution, which was eventually found by identifying all columns which share the same coordinates, adding their corresponding axial stresses, and saving the values of the moments in both directions to be accounted for the verification. The tests after this change show that it significantly improved the quality of the 2D frame modelling, since it got very close results to the same structure modelled as a 3D Frame. Comparing the bottom section of the corner column of both models:

Table 6.2 – Axial force and bending moments acting on a corner column due to the static load.

Static Response		
	2D Frame	3D Frame
N [N]	-240553,68	-243810,24
M2 [KNm]	4701,79	4868,34
M3 [KNm]	-4696,73	-4863,92

Table 6.3 – Absolute values of stress resultants acting on a corner column due to the dynamic load.

Dynamic Response		
	2D Frame	3D Frame
|N| [N]	226350,52	226350,52
|M2| [KNm]	182974,40	182974,40
|M3| [KNm]	183258,03	183258,03

As can be seen from the previous tables, the only difference between the two models for this element resides in the static response, with the values corresponding to the 3D frame being slightly higher. This happens because the axial stiffness on the columns of the 2D frame is doubled, which leads to the increase of the axial force in the most loaded (central) columns, and a decrease in the axial force for the remaining peripheral and less loaded columns.

The following values of ω were computed for the base cross-section of the same column:

Table 6.4 – Estimation of the mechanical reinforcement ratio for both models.

Mechanical reinforcement ratio	
2D	3D
0,3044615	0,304056141

This difference is small enough to be disregarded, and it was concluded that most of the models with frames can be achieved using only 2D frames. These allow for more flexibility regarding the overall design, coming only at the cost of more input data.

6.2 Connected Core Walls vs. Regular Walls

In order to estimate the differences between the models with connected walls and regular walls (see Sections 3.3.1 and 3.3.2), the two were tested against a continuous beam model. Three cases were considered: an I-section core only with regular walls meeting at nodes, another I-section this time with the web connecting directly to the flanges, and finally the continuous beam (simulated as a 3D Frame element with a single beam) with the same cross-section as that of the core being modelled:

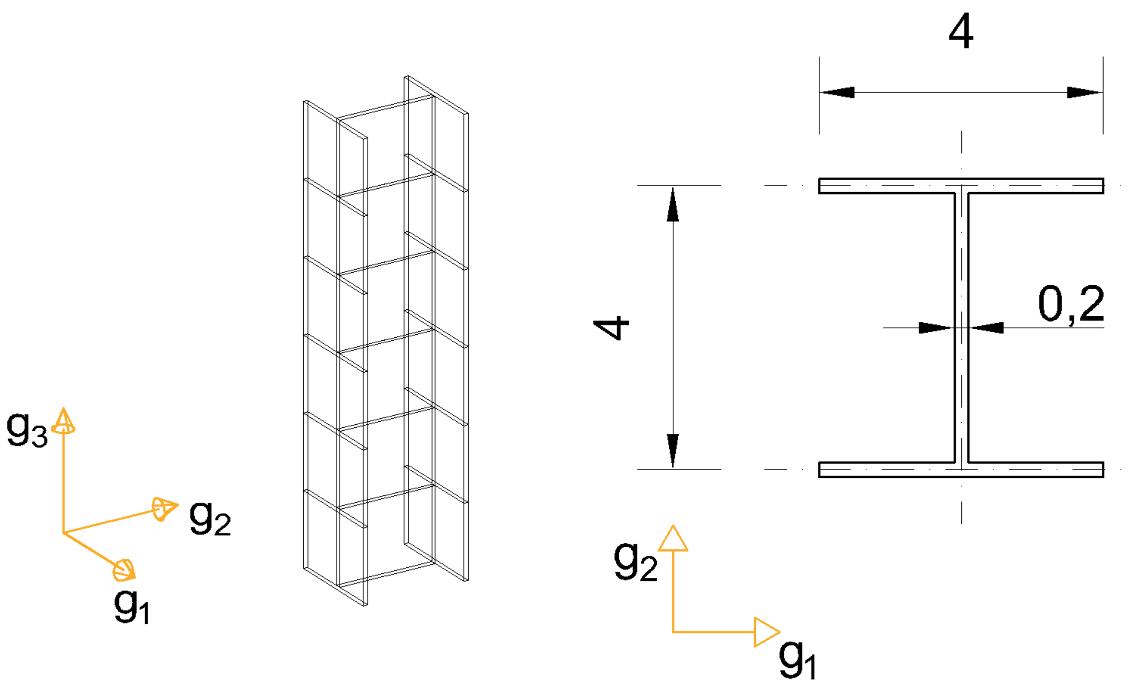


Figure 6.2 – Details of the wall being modelled.

The model has 6 stories, each with a square slab with 6 meters of side length (not shown in the previous figure for clarity).

The following tables summarize the dynamic behavior of the three models:

Table 6.5 – Effective masses and frequencies for modes in the x direction for all three models

X Direction	Effective Mass			Frequency [Hz]		
Mode	Regular Wall	Connected Wall	Column Model	Regular Wall	Connected Wall	Column Model
1	67,34%	66,72%	66,72%	3,370	3,417	3,417
2	21,33%	20,35%	20,35%	19,771	21,705	21,705
3	7,04%	6,96%	6,96%	49,621	61,391	61,391
4	2,95%	3,46%	3,46%	83,325	120,424	120,424
5	1,09%	1,84%	1,84%	113,523	192,985	192,985
6	0,25%	0,67%	0,67%	134,164	257,651	257,651

Table 6.6 – Effective masses and frequencies for modes in the y direction for all three models

Y Direction	Effective Mass			Frequency [Hz]		
Mode	Regular Wall	Connected Wall	Column Model	Regular Wall	Connected Wall	Column Model
1	68,26%	67,92%	66,72%	6,157	6,220	6,393
2	21,75%	21,90%	20,35%	32,603	34,189	40,606
3	6,28%	6,70%	6,96%	75,051	80,286	114,852
4	2,42%	2,48%	3,46%	118,806	127,195	225,293
5	0,99%	0,82%	1,84%	158,575	166,146	361,041
6	0,29%	0,17%	0,67%	188,536	191,375	482,020

As could be expected, the differences between the connected walls and the column model for the modes along direction X are very small, both for the effective masses and the frequencies, while the regular walls, being more flexible, have bigger differences to the other two models. This suggests that for this direction – the direction of least inertia of the I cross-section - the connected walls method does indeed model an I-shaped core better than its previously defined alternative.

When considering the Y direction, both wall models differ significantly from the column. An analysis of the frequencies shows that the connected walls method gives slightly better results than its alternative. This was also expected as the stiffness of the core wall should be bigger when the “flanges” are considered as continuous pieces instead of being separated in half; and since the mass is the same for both cases, the one with higher stiffness is the one with higher frequencies. The remaining difference between the connected walls and the column model is attributed to the fact that the wall model connects the walls only at the level of the floors, while a column beam has the web and flanges continuously connected, giving it an even higher stiffness, and thus higher frequencies. However, this effect only significant in higher modes, which are not so important for the analysis at hand.

As for the torsional modes, the results for the two types cores are given in the following table (effective masses are not presented since they are zero for these modes):

Table 6.7 – Frequencies of the rotational modes for the three models

Rotational Modes			
	Frequency (Hz)		
Mode	Regular Wall	Connected Wall	Column Model
1	2,752	2,790	0,966
2	16,143	17,722	2,841
3	40,516	50,125	4,551
4	68,035	98,326	5,996
5	92,692	157,571	7,093
6	109,544	210,371	7,778

Once again, the connected walls model was expected to have higher frequencies than the regular walls since it is slightly stiffer. The column model in this case is not appropriate, since it fails to model restrained warping effects, particularly important for this shape of cross-section.

6.3 Asymmetric and irregular buildings

One of the characteristics desired for the program was that it should be able to model irregular and asymmetric buildings. This can easily be achieved by building the model using 2D frames, which gives more flexibility for creating structures that vary in height. An example of this is given below, further completed by the addition of a core wall, in order to create a bigger asymmetry and to study the interaction between the two types of sub-structures as well as the effect of the increase in the global stiffness of the model.

The following structure was modelled:

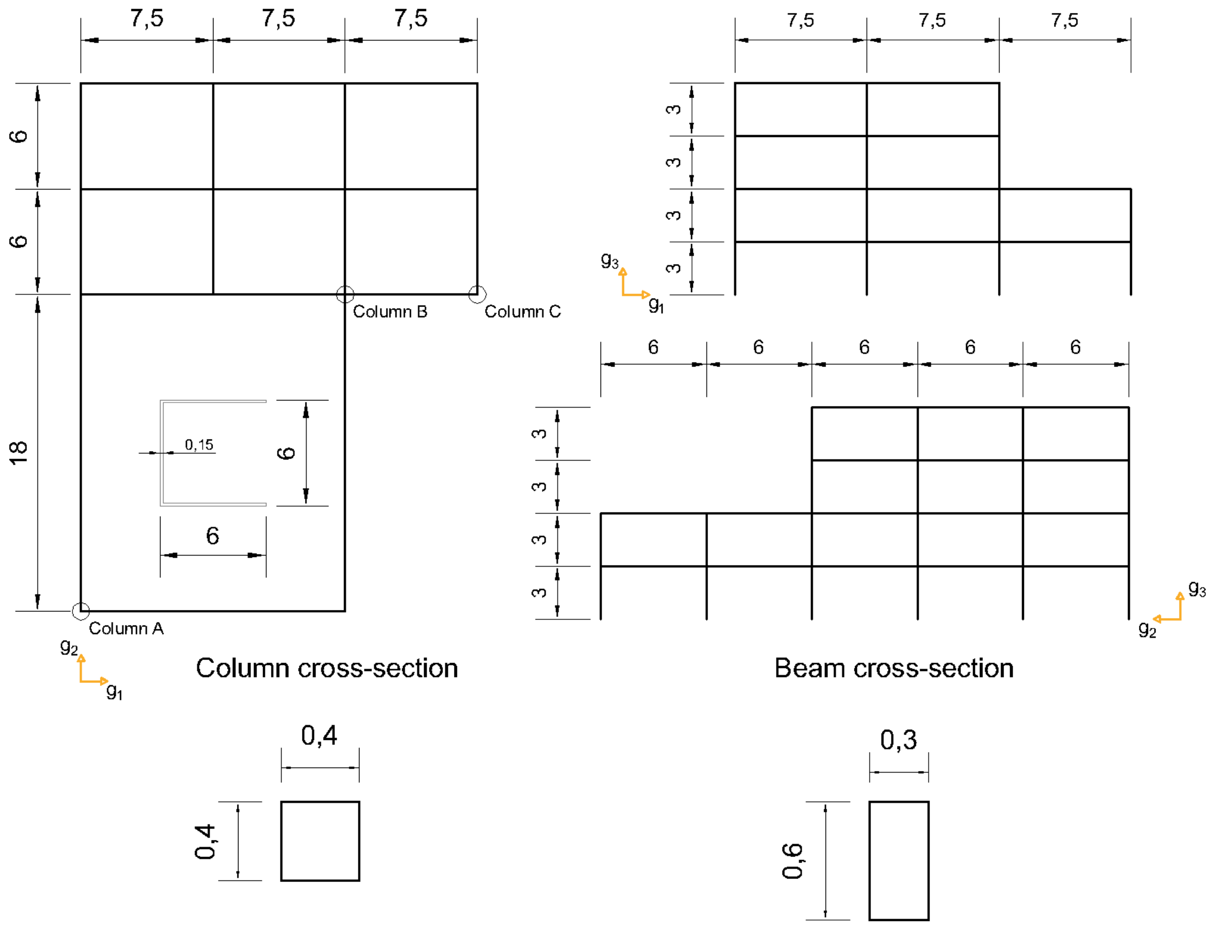


Figure 6.3 – Details of the model.

For illustration, the first three modes are represented in plan view in Figure 6.4. To make these drawings, one first obtains the modal center of rotation of each floor, and then plots the rotation around it.

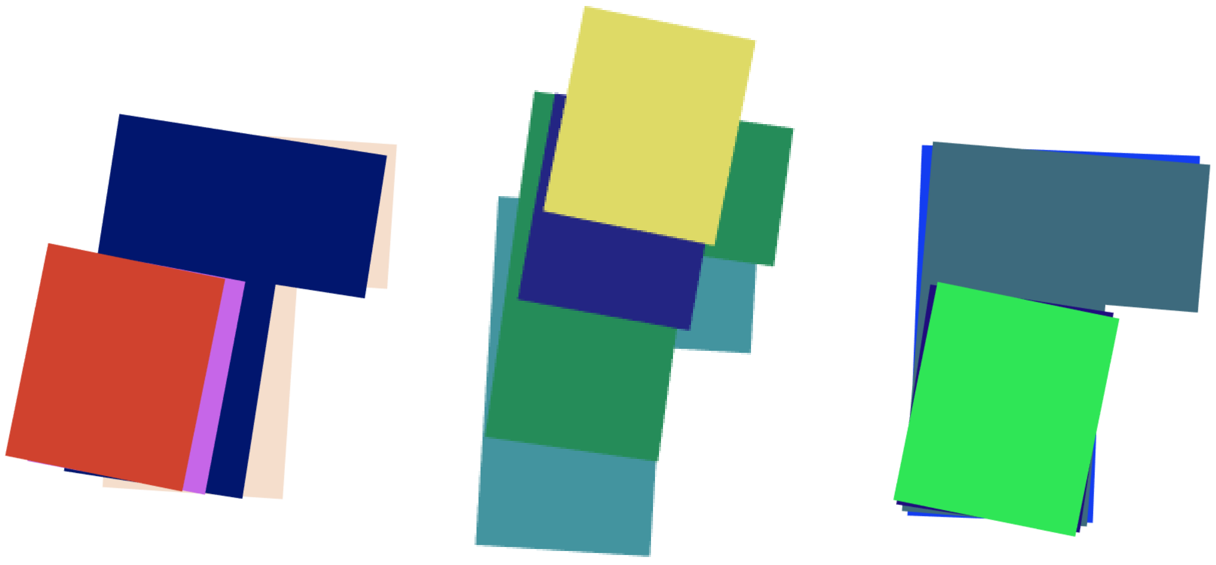


Figure 6.4 – Plan view of the first three modes of the model.

Initially, only considering the 2D frames, the model gives high values for the mechanical reinforcement ratio in the corner columns of the taller part of the structure (element A in Figure 6.1), on the column directly above the setback (element B) and on some of the columns in the lower part of the structure (of which element C is an example):

Table 6.8 – Maximum values of $\boldsymbol{\omega}$ in the model

Column	Mechanical reinforcement ratio
A	0,401
B	0,381
C	0,373

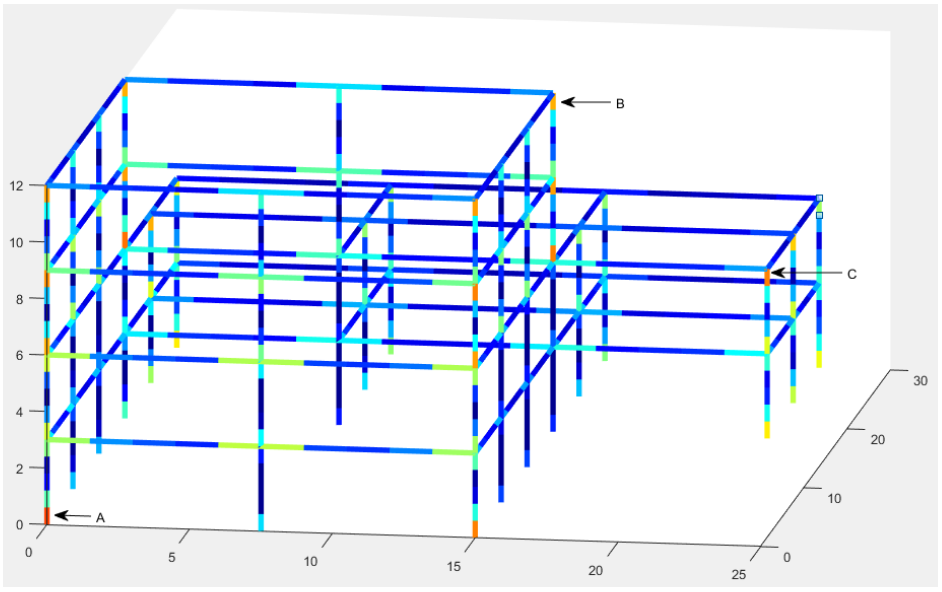


Figure 6.5 – Program plot showing the levels of stress.

The following figure displays the output plot showing clearly the most critical elements, which are the ones that do not verify the conditions defined in Section 4.3:

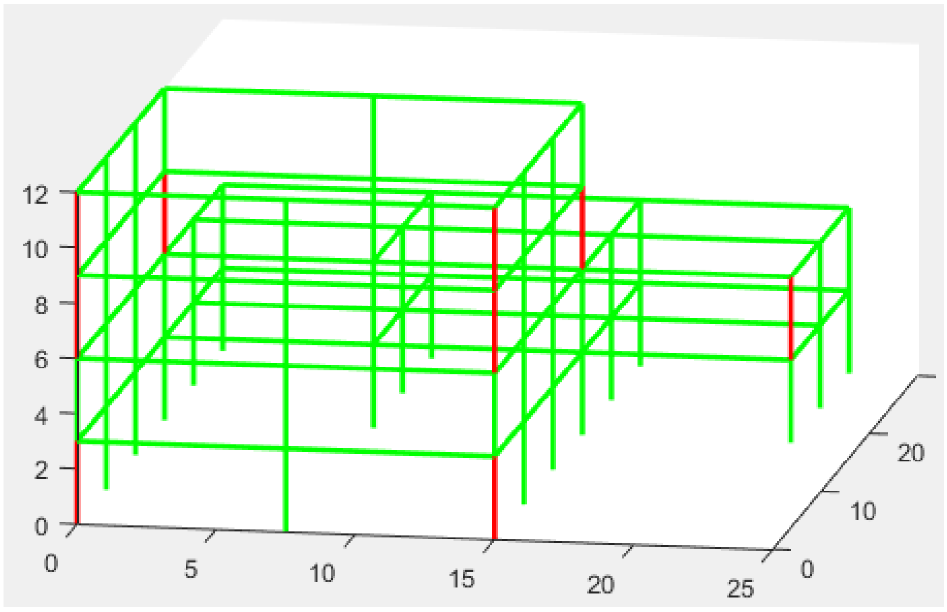


Figure 6.6 – Program plot showing over-stressed elements.

When considering a U-section core wall in the taller part of the model with the dimensions detailed in Figure 6.3, the mechanical reinforcement ratio dropped significantly on every element:

Table 6.9 – Maximum values of the mechanical reinforcement ratio in both types of sub-structures

Maximum values of ω after adding a core wall	
Columns	Core
0,173	0,171

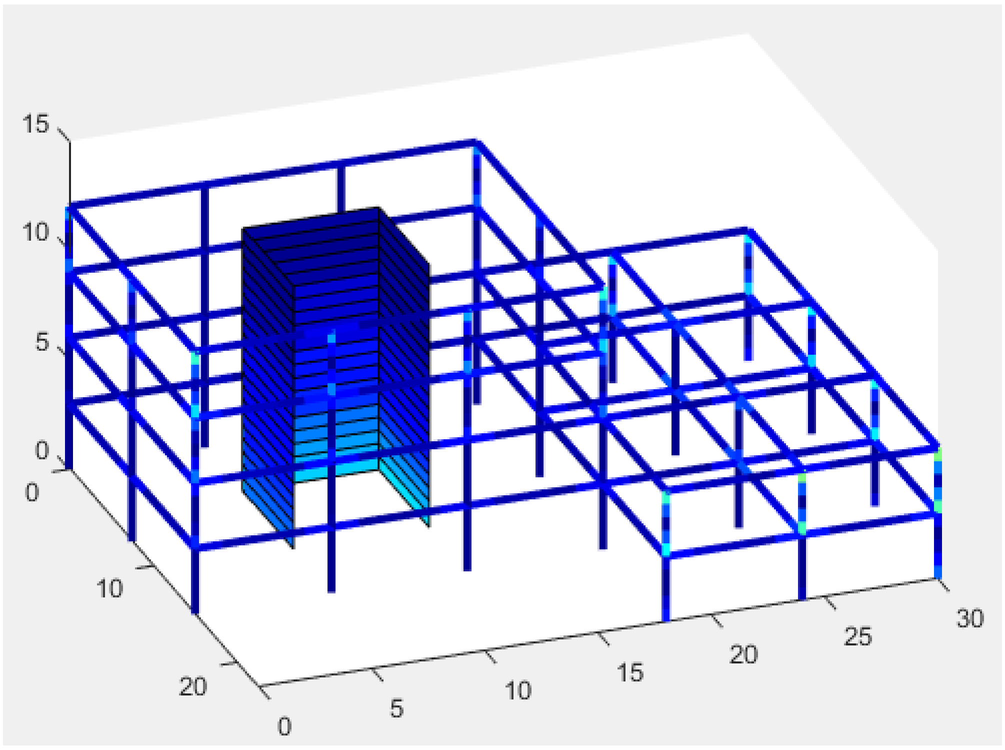


Figure 6.7 – Levels of stress after adding the core wall to the model.

The addition of the wall is also reflected on the frequencies of the model:

Table 6.10 – Modal frequencies registered for both models

	Frequency (Hz)	
Mode	Without Core Wall	With Core Wall
1	1,106	2,683
2	1,470	5,682
3	2,351	7,007
4	2,815	9,380
5	3,345	18,378
6	4,210	28,114
7	5,460	35,724
8	6,476	35,899
9	7,085	65,394
10	7,963	85,634
11	9,060	98,660
12	10,971	131,301

6.4 Combination of the mechanical reinforcement ratio

A better alternative to the analysis as presented in Section 4.3 consists in calculating the linear contribution for the mechanical reinforcement ratio mode by mode, and only then doing the modal and directional combinations. This could lead to much more efficient design since much of the information about the interaction of stresses is lost when doing the combinations beforehand. It is important to stress that in this new approach the modal internal forces still retain their sign correlation. Hence, both positive

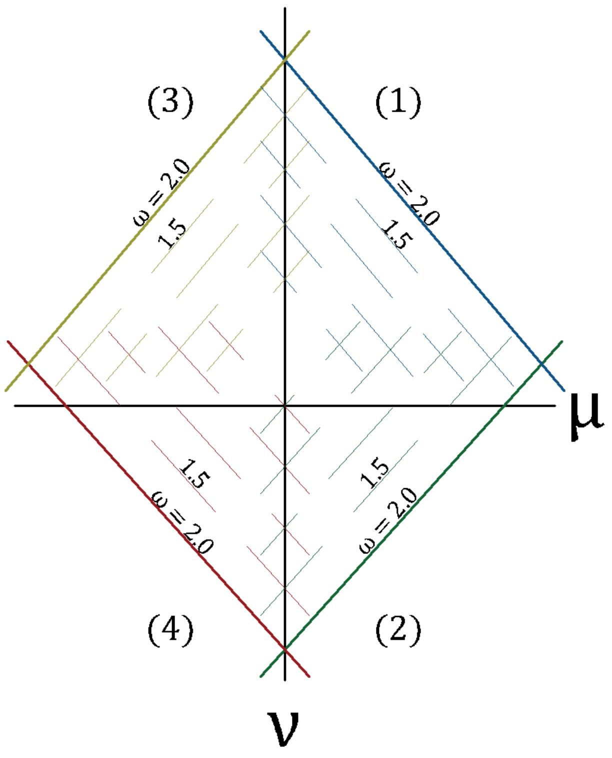


Figure 6.8 – Example of a discretization using four planes.

and negative moments should be considered, which means that four families of planes have to be taken into account instead of two (as shown in Figure 6.8).

The remaining two planes can easily be retrieved, since the effect of the moment is symmetric. Thus, they are derived from the previous two by taking each of them and changing the sign of the first component of the vector $\vec{t}$. For instance, looking back at the diagram which was discretized in Section 4.3.3, the four families become:

$$\begin{cases} \omega^{(1)} = \vec{x}\cdot\vec{t}^{(1)} + \mathrm{C}^{(1)} = 2.375\mu - 1.0\nu - 0.85 \\ \omega^{(2)} = \vec{x}\cdot\vec{t}^{(2)} + \mathrm{C}^{(2)} = 2.222\mu + 1.0\nu + 0 \\ \omega^{(3)} = \vec{x}\cdot\vec{t}^{(3)} + \mathrm{C}^{(3)} = -2.375\mu - 1.0\nu - 0.85 \\ \omega^{(4)} = \vec{x}\cdot\vec{t}^{(4)} + \mathrm{C}^{(4)} = -2.222\mu + 1.0\nu + 0 \end{cases}$$

where $\vec{x}$ remains a vector on the (μ, ν) plane.

For each family f of planes defined by a vector $\vec{t}^{(f)}$ and scalar $C^{(f)}$, instead of having equation (4.31) and using it with the values obtained from the presented combinations, one could instead compute the contribution to the value of ω due to every i^{th} mode, $\omega'^{(f)}_i$, by means of the linear expression:

$$\omega'^{(f)}_i = \vec{x}_i \cdot \vec{t}^{(f)} \tag{6.1}$$

Where $\vec{x}_i$ encodes the values of the relative stresses due to mode i. The result of this expression is not the real value of the mechanical reinforcement ratio needed due to mode n, since the constant $C^{(f)}$ from equation (4.31) has not been included. The reason for doing this is that by taking advantage of the linearity of the previous expression, a combination for each direction of the seismic action can be made:

$$\begin{cases} \omega'^{(f)}_x = \mathrm{CQC}\left(\omega'^{(f)}_{1,x}, \dots, \omega'^{(f)}_{n,x}\right) \\ \omega'^{(f)}_y = \mathrm{CQC}\left(\omega'^{(f)}_{1,y}, \dots, \omega'^{(f)}_{n,y}\right) \end{cases}$$

with the operator CQC standing for the complete quadratic combination presented in Section 4.2.3, using the same modal correlations. The full dynamic contribution is then retrieved through a SRSS combination:

$$\omega'^{(f)}_d = \mathrm{SRSS}\left(\omega'^{(f)}_x, \omega'^{(f)}_y\right) = \sqrt{\left(\omega'^{(f)}_x\right)^2 + \left(\omega'^{(f)}_y\right)^2}$$

For the case of the static load, again through equation (6.1) an analogous value ω'_s can be found for the ratio due to the static stresses:

$$\omega'^{(f)}_s = \vec{x}_s \cdot \vec{t}^{(f)}$$

An estimate for the value of the mechanical reinforcement ratio is arrived at by adding the static and dynamic components to the constant $C^{(f)}$:

$$\omega^{(f)} = \omega'^{(f)}_s + \omega'^{(f)}_d + C^{(f)} \tag{6.2}$$

Having completed this calculation for all families of planes, the maximum among them is taken.

$$\omega = \max_f \left\{\omega^{(f)}\right\}$$

As an example, a detailed analysis is made to give a general idea of this procedure, where for simplicity, only the seismic action acting along the y direction (corresponding to axes $\overrightarrow{g_2}$ in Figure 6.1) is considered. As such, there is no need to perform a combination of both directions, and thus the value taken for the dynamic contribution, $\omega'^{(f)}_d$, is simply that of $\omega'^{(f)}_y$.

Considering once again the interaction diagram from Section 4.3.3 and its discretization (see Figure 4.3), one can perform the computation of ω_i' for each plane. Taking the example of the 3D Frame presented in Section 6.1 and analyzing the bottom section of a corner column (all four of them being identical), the following table shows the axial stress and the moment in the direction $\overrightarrow{l_2}$ of its local frame, as well the resulting value for ω found using the standard method presented in Section 4.3.2:

Table 6.11 – Results from the analysis for the bottom of a corner column

N (N)	M2 (Nm)	ν	μ	ω
-82868	188126	-0,017	0,098	0,200

Alternatively, looking at the modal stresses in the y direction and using equation (6.1) for all families of planes (only the modes with non-zero participation factors along each direction are presented):

Table 6.12 – Values of $\boldsymbol{\omega}'$ for all modes and all families of planes

Results for the seismic action along direction y						
			Family of planes			
Mode	ν	μ	$\omega'^{(1)}$	$\omega'^{(2)}$	$\omega'^{(3)}$	$\omega'^{(4)}$
1	0,033	-0,094	-0,257	-0,175	0,190	0,242
4	-0,002	-0,015	-0,034	-0,036	0,039	0,032
7	0,000	-0,005	-0,013	-0,012	0,012	0,012
10	0,000	-0,002	-0,004	-0,003	0,004	0,003

After applying the modal combination:

Table 6.13 – Linear contribution to the mechanical reinforcement ratio due the dynamic response

Dynamic contribution			
$\omega'^{(1)}_d$	$\omega'^{(2)}_d$	$\omega'^{(3)}_d$	$\omega'^{(4)}_d$
0,259	0,180	0,194	0,245

Now, for the static response, equation (6.1) yielded the following results:

Table 6.14 – Linear contribution to the mechanical reinforcement ratio of the static response

Static contribution			
$\omega_s'^{(1)}$	$\omega_s'^{(1)}$	$\omega_s'^{(1)}$	$\omega_s'^{(1)}$
0,057	-0,045	0,045	-0,056

Finally, using equation (6.2), ω is given for each family of planes by:

Table 6.15 – Results for the four families of planes

$\omega^{(1)}$	$\omega^{(2)}$	$\omega^{(3)}$	$\omega^{(4)}$
-0,534	0,135	-0,611	0,188

Taking the maximum among them gives $\omega = \omega^{(4)} = 0{,}188$ which is a smaller figure than the one arrived at earlier, which suggests this method could improve the efficiency of the design by giving less conservative, more accurate results.

Note however that for the application of this methodology to the design of columns subjected to biaxial bending with axial force, one needs first to discretize the full 3D interaction diagram.

The following figure illustrates the differences between the two methods, with the first result in purple and the result from the combination of the mechanical ratios in red.

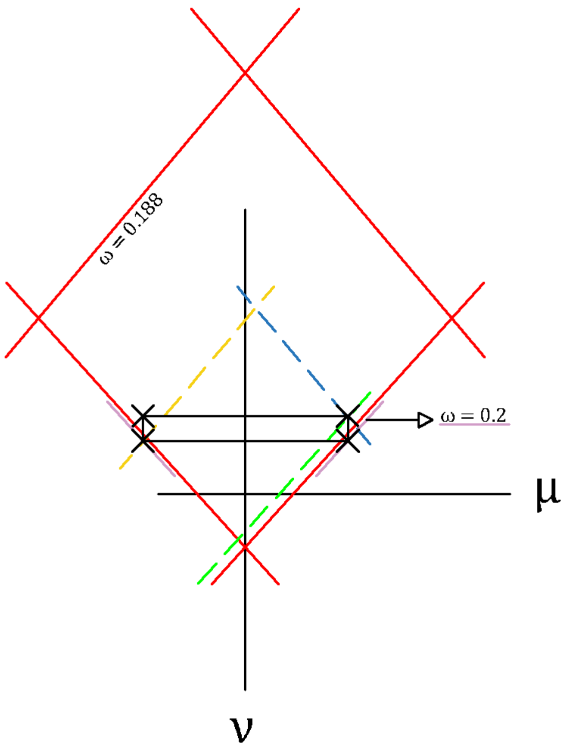


Figure 6.9 – Comparison of results from both methods.

7 CONCLUSION

The objective of the dissertation was to implement a program which could do a simple but reliable analysis of the structural behavior of a given model, making use of a few key simplifications.

The concept of three degrees of freedom per story was introduced with the aim of making the dynamic analysis less demanding, since that is an important part of the structural design of buildings. It proved to be a reasonable simplification, since most of the dynamic effects were captured in the models which were tested. The division into sub-structures was an important decision as well, since it allowed for an intuitive way of conceiving and modelling a structure, which can often be the most complicated part when working with a structural analysis software. This was an aspect which was much sought-after, since the program is intended to be used as an aid to conceptual design.

After having made these decisions, the employment of the notion of “Slave” and “Local” degrees of freedom was key, as it allowed to focus merely on the degrees of freedom which are directly dependent on the global degrees of freedom introduced earlier, once again reducing the complexity of the analysis.

A method of modelling walls was introduced, which despite having a very simple formulation, was able to model some complex behaviors, such as restrained warping. This helped to enlarge the possibilities for the modelling, thus making the program applicable to a much wider variety of situations.

The method for discretizing the axial force/bending moment interaction diagrams greatly improved the capabilities of the software, which when equipped with it was able to perform an evaluation of over-stressed elements in a swift manner. This was one of the main points of the work, since this type of information may not be immediately obvious when conceiving a building, yet it can greatly influence the outcome of the project later on. The results of the analysis made in Section 6.4 suggest that this method could be further improved to give more accurate results.

The introduction of graphic output in the form of plots was also quite helpful, since it allows for information to be conveyed in a much more direct fashion than by outputting arrays or tables with numeric values. This also adds to the ease of use of the program, which was the main overarching principle throughout its conception.

A case study was presented, and although it was not fully explored it has shown the potential of the proposed approach.

7.1 Future developments

Several things can be improved upon. The modelling of 3D Frames only allows for very regular structures without setbacks or unusual features. This is a drawback since the process of inputting all the

2D frames that belong to a building can be tedious, and thus a generalization of irregular frames from 2D to 3D without too many extra input parameters would be a good accomplishment, and would further increase the accessibility of the program.

One very interesting extension to the work presented in this document would be the introduction of a routine which instead of taking the pre-defined cross-sections would be able to give a minimum bound for the area of the cross-section (or one of the dimensions, with the other being fixed) of each element. This would minimize the amount of input data and could potentially allow for even more efficient design. Other relevant improvements for the handling of cross-sections would be the generalization from rectangular to any type of solid or thin-walled cross-sections, as well as the consideration of steel structures.

Another possible improvement would be to discard the assumption that walls behave as linear elements and instead allow for their modeling as “3D shell” elements. This could reduce quite significantly the differences between the I-section core wall and the 3D beam model with the same cross-section which were seen in Section 6.2. However, increasing the complexity of the structural modelling goes against the philosophy of the present approach, being more appropriate for the final stages of design.

Finally, the software could be used to perform several kinds of parametric studies wherein one parameter is made to vary, while the rest of the structure is fixed, in an attempt to estimate what would be the best possible value for that parameter. For instance, in a project with both a frame and a core wall, such a study could be made by firstly defining the frame and then sequentially running the program while incrementing the position of the core wall in one direction. By comparing the results of each analysis, an estimate for the best location for the core wall within the model would quite probably be found.

BIBLIOGRAPHY

[1] Delgado, J. M., Hofmeyer, H., *Automated generation of structural solutions based on spatial designs*, Automation in Construction, 35: 528-541, 2013.

[2] Rolvink, A., Mueller, C. and Coenders, J., *State on the Art of Computational Tools for Conceptual Structural Design*, Proceedings of the IASS-SLTE 2014 Symposium “Shells, Membranes and Spatial Structures: Footprints”, Brasil, 2014.

[3] *Eurocódigo NP EN1998–1:2010 – Projecto de Estruturas para resistência aos sismos. Parte 1: Regras gerais, acções sísmicas e regras para edifícios*, 2010.

[4] Clough, R.W.and Penzien, J., *Dynamics of Structures*, 2nd Edition, McGraw-Hill, New York, 1993.

[5] Grupo de Betão Armado e Pré-Esforçado, *Estruturas de Betão I – Folhas de Apoio às Aulas*, Coordenação: José Noronha da Câmara, IST, Lisboa, 2014.

[6] Grupo de Betão Armado e Pré-Esforçado, *Estados Limites Últimos – Flexão Composta de Secções Rectangulares*, IST, Lisboa, 2010.

[7] Tomaz, A. R., *Seismic Analysis of Structures: Stress-resultant Interaction based on Response Spectra*, MSc thesis, IST, Lisboa, 2017.

[8] Menun, C. and Der Kiureghian, A., *Envelopes for seismic response vectors. I: Theory*., Journal of Structural Engineering, 126(4):467-473, 2000.

[9] Menun, C. and Der Kiureghian, A., *Envelopes for seismic response vectors. II: Application*., Journal of Structural Engineering, 126(4):474-481, 2000.

[10] The Mathworks Inc. MATLAB R2018b, Massachusetts, United States of America.

Appendix A – Structure Class Code

```
classdef Estr
    properties
        Porticos
        Porticos3D
        Nucleos
        Pisos
        Massa
        KGG
        FG
        MGG
        Modos
        Freq
        Part
        Meff
        Miu_CQC
    end
    methods
        function obj =
Estr(VectorPortico,VectorPortico3D,VectorNucleo,AltPisos,Lajes,Sismica,Graficos)

            obj.Porticos = VectorPortico; %%% Guardar as sub-estruturas
            obj.Porticos3D = VectorPortico3D;
            obj.Nucleos = VectorNucleo;

            obj.Pisos = AltPisos;
            PisosSup = AltPisos(AltPisos>0); %% Lista dos pisos com cota > 0

            obj.KGG = zeros(3 * length(PisosSup)); %% Inicialização
            obj.FG = zeros(3 * length(PisosSup), 1);
            obj.MGG = zeros(3 * length(PisosSup));
            obj.Modos = zeros(3 * length(PisosSup));
            obj.Freq = zeros(3 * length(PisosSup), 1);

            PilaresPorticos = 0;
            for ip = 1:length(obj.Porticos)
                PilaresPorticos = PilaresPorticos + obj.Porticos(ip).nPilares;
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%     KGG e MGG     %%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            for ip = 1:length(obj.Porticos) %%%%%%% Condensação e Matriz KGG, contribuiçao dos
porticos
                obj.KGG = obj.KGG + obj.Porticos(ip).KGG;
                obj.FG = obj.FG + obj.Porticos(ip).FG;
            end

            for ip = 1:length(obj.Porticos3D) %%%%%%%%% Matriz KGG, contribuiçao dos porticos
3D
                obj.KGG = obj.KGG + obj.Porticos3D(ip).KGG;
                obj.FG = obj.FG + obj.Porticos3D(ip).FG;
            end

            for in = 1:length(obj.Nucleos) %%% Matriz KGG, contribuiçao dos nucleos
                obj.KGG = obj.KGG + obj.Nucleos(in).KGG;
                obj.FG = obj.FG + obj.Nucleos(in).FG;
            end

            obj.Massa = 0;

            for il = 1:length(Lajes) %%% Matriz de Massas
                M = Lajes(il).M;
                Sx = M * Lajes(il).cg(2);
                Sy = M * Lajes(il).cg(1);
                Ip = M * ((Lajes(il).lx^2 + Lajes(il).ly^2)/12 + sum(Lajes(il).cg.^2));
                Mlaje = [M,    0,    -Sx;
                    0,    M,     Sy;
                    -Sx,   Sy,    Ip];
                ind = (1:3) + 3*(find(AltPisos==Lajes(il).h) - 2);
                obj.MGG(ind,ind) = obj.MGG(ind,ind) + Mlaje;
                obj.Massa = obj.Massa + M; %%%%% calcular a massa total
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%     Estática      %%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            q_est = obj.KGG \ obj.FG;
            for ip = 1:length(obj.Porticos) %%% Calcular esforços nos porticos
                obj.Porticos(ip) = Calc_Esf(obj.Porticos(ip), q_est, true);
                obj.Porticos(ip).Esf_Estatica = obj.Porticos(ip).Esf;
            end
            for ip = 1 : length(obj.Porticos3D) %%% Calcular esforços nos porticos 3D
                obj.Porticos3D(ip) = Calc_Esf(obj.Porticos3D(ip), q_est, true);
                obj.Porticos3D(ip).Esf_Estatica = obj.Porticos3D(ip).Esf;
            end
            for in = 1:length(obj.Nucleos) %%% Calcular esforços nos nucleos
                obj.Nucleos(in) = Calc_Esf(obj.Nucleos(in), q_est, true);
                obj.Nucleos(in).Esf_Estatica = obj.Nucleos(in).Esf;
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%   Modos e Freq.   %%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            [obj.Modos, Freq2] = eig(obj.KGG,obj.MGG);
            obj.Freq = diag(Freq2).^(1/2)/(2*pi); %%%%%%% frequencias em Hz
            [obj.Freq, I] = sort(obj.Freq);
            obj.Modos = obj.Modos(:,I);

            for im = 1:length(obj.Modos) %%%%%%%%% Normalizar com respeito às massas
                obj.Modos(:,im) = obj.Modos(:,im) / sqrt(obj.Modos(:,im)' * obj.MGG *
obj.Modos(:,im));
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%   Participações   %%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            obj.Part = zeros(3,length(obj.Modos)); %%%% Participações modais
            u_x = repmat([1; 0; 0], length(PisosSup), 1);
            u_y = repmat([0; 1; 0], length(PisosSup), 1);

            obj.Part = obj.Modos' * obj.MGG * [u_x, u_y]; %%%%% Fazer as combinaçoes para ter
os esf.

            obj.Meff = zeros(size(obj.Part));

            for im = 1 : length(obj.Modos)
                obj.Meff(im,:) = obj.Part(im,:).^2/obj.Massa;
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%% Analise Espectral %%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            T_modos = obj.Freq.^-1;
            S_modos = zeros(size(T_modos));
            T_B = .1;
            Tabela_T_C = [.6, .25; .6, .25; .6, .25; .8, .3; .6, .25];
            T_D = 2;
            switch Sismica.Zona
                case 'A'
                    T_C = Tabela_T_C(1,Sismica.Tipo);
                case 'B'
                    T_C = Tabela_T_C(2,Sismica.Tipo);
                case 'C'
                    T_C = Tabela_T_C(3,Sismica.Tipo);
                case 'D'
                    T_C = Tabela_T_C(4,Sismica.Tipo);
                case 'E'
                    T_C = Tabela_T_C(5,Sismica.Tipo);
            end

            for im = 1 : length(obj.Modos)
                if T_modos(im) > T_D
                    S_modos(im) = Sismica.ag_x_S * 2.5 / Sismica.q * T_C * T_D /
T_modos(im)^2;
                elseif T_modos(im) > T_C
                    S_modos(im) = Sismica.ag_x_S * 2.5 / Sismica.q * T_C / T_modos(im);
                elseif T_modos(im) > T_B
                    S_modos(im) = Sismica.ag_x_S * 2.5 / Sismica.q;
                else
                    S_modos(im) = Sismica.ag_x_S * (2/3 + T_modos(im) / T_B * (2.5 / Sismica.q
- 2/3));
                end
            end

            q_spec = zeros([size(obj.Modos), 2]); %%%%%%%% Deslocamentos espectrais (cada
coluna é um modo), a 3a dim e para a direcçao do sismo
            for idir = 1:2
```

```
                for im = 1:length(obj.Modos)
                    q_spec(:,im,idir) = obj.Modos(:,im) * obj.Part(im,idir) * S_modos(im) / (4
* pi^2 * obj.Freq(im)^2);
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%    Esf. Modais    %%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            for ip = 1:length(obj.Porticos)
                obj.Porticos(ip).Esf_Modais = struct('Esf', cell(length(obj.Modos),2));
            end
            for ip = 1:length(obj.Porticos3D)
                obj.Porticos3D(ip).Esf_Modais = struct('Esf', cell(length(obj.Modos),2));
            end
            for in = 1:length(obj.Nucleos)
                obj.Nucleos(in).Esf_Modais = struct('Esf', cell(length(obj.Modos),2));
            end

            for idir = 1:2
                for ip = 1:length(obj.Porticos) %%% Calcular esforços nos porticos
                    for im = 1:length(obj.Modos)
                        obj.Porticos(ip) = Calc_Esf(obj.Porticos(ip), q_spec(:,im,idir),
false); %%Mudar para devolver so os esf?
                        obj.Porticos(ip).Esf_Modais(im,idir).Esf = obj.Porticos(ip).Esf;
                    end
                end
                for ip = 1 : length(obj.Porticos3D) %%% Calcular esforços nos porticos 3D
                    for im = 1:length(obj.Modos)
                        obj.Porticos3D(ip) = Calc_Esf(obj.Porticos3D(ip),q_spec(:,im,idir),
false); %%Mudar para devolver so os esf?
                        obj.Porticos3D(ip).Esf_Modais(im,idir).Esf = obj.Porticos3D(ip).Esf;
                    end
                end
                for in = 1:length(obj.Nucleos) %%% Calcular esforços nos nucleos
                    for im = 1:length(obj.Modos)
                        obj.Nucleos(in) = Calc_Esf(obj.Nucleos(in), q_spec(:,im,idir), false);
                        obj.Nucleos(in).Esf_Modais(im,idir).Esf = obj.Nucleos(in).Esf;
                    end
                end
            end

            obj.Miu_CQC = zeros(size(obj.Modos)); %%%%%% Correlações CQC
            for im = 1 : length(obj.Modos)
                for ig = 1 : length(obj.Modos)
                    r = obj.Freq(im)/obj.Freq(ig);
                    obj.Miu_CQC(im,ig) = ( 8 * Sismica.amort ^ 2 * (1 + r) * r ^(3/2) )/( (1-
r^2)^2 + 4 * Sismica.amort ^ 2 * r * (1+r)^2 );
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%    Combinações    %%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for ip = 1:length(obj.Porticos)
                obj.Porticos(ip) = Comb(obj.Porticos(ip),obj.Miu_CQC);
            end
            for ip = 1:length(obj.Porticos3D)
                obj.Porticos3D(ip) = Comb(obj.Porticos3D(ip),obj.Miu_CQC);
            end
            for in = 1:length(obj.Nucleos)
                obj.Nucleos(in) = Comb(obj.Nucleos(in),obj.Miu_CQC);
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%   Juntar Pilares   %%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            Esf_Pilares_Est = struct('N',cell(PilaresPorticos,1),'M',[],'V',[]); %%%% Estatica
            Esf_Pilares_Din = struct('N',cell(PilaresPorticos,1),'M',[],'V',[]); %%%% Dinamica
            Pos_Pilares = zeros(6,PilaresPorticos);
            Pilares_glob = struct('ind',cell(PilaresPorticos,1),'dir',[0, 0]);
            indb = 1;
            cont = 1;
            for ip = 1:length(obj.Porticos)
                Esf_Pilares_Est(indb:(indb + obj.Porticos(ip).nPilares - 1)) =
obj.Porticos(ip).Esf_Estatica(1:obj.Porticos(ip).nPilares);
                Esf_Pilares_Din(indb:(indb + obj.Porticos(ip).nPilares - 1)) =
obj.Porticos(ip).Comb_Final(1:obj.Porticos(ip).nPilares);
                for ib = 1 : obj.Porticos(ip).nPilares %%% guardar as posiçoes
                    pos1 = obj.Porticos(ip).Nos(obj.Porticos(ip).Barras(ib).ni).x;
                    pos2 = obj.Porticos(ip).Nos(obj.Porticos(ip).Barras(ib).nf).x;
                    Pos_Pilares(1:3,indb + ib - 1) =[pos1(1) * obj.Porticos(ip).dir' +
obj.Porticos(ip).origem'; pos1(2)];
                    Pos_Pilares(4:6,indb + ib - 1) =[pos2(1) * obj.Porticos(ip).dir' +
obj.Porticos(ip).origem'; pos2(2)];
                    for ib2 = 1 : indb %% Comparar
                        if all(Pos_Pilares(:,indb + ib - 1) == Pos_Pilares(:,ib2))
                            Pilares_glob(indb + ib - 1).ind = ib2;
                            break
                        end
                    end
                    if ib2 == indb
                        Pilares_glob(indb + ib - 1).ind = cont;
                        cont = cont + 1;
                    end
                    Pilares_glob(indb + ib - 1).dir = obj.Porticos(ip).dir;
                end
                indb = indb + obj.Porticos(ip).nPilares;
            end
            nPilares_j = cont - 1;

            Esf_Pilares_N_Est = zeros(nPilares_j,21);
            Esf_Pilares_M3_Est = zeros(PilaresPorticos,21);
            Esf_Pilares_V2_Est = zeros(PilaresPorticos,21);

            Esf_Pilares_N_Din = zeros(nPilares_j,21);
            Esf_Pilares_M3_Din = zeros(PilaresPorticos,21);
            Esf_Pilares_V2_Din = zeros(PilaresPorticos,21);

            % struct('N',cell(nPilares_j,1),'M',zeros(1,21),'V',zeros(1,21));

            for ib = 1:length(Pilares_glob) %%% Atribuir os esforços
                Esf_Pilares_N_Est(Pilares_glob(ib).ind,:) =
Esf_Pilares_N_Est(Pilares_glob(ib).ind,:) + Esf_Pilares_Est(ib).N;
                Esf_Pilares_N_Din(Pilares_glob(ib).ind,:) =
(Esf_Pilares_N_Din(Pilares_glob(ib).ind,:).^2 + Esf_Pilares_Din(ib).N.^2).^(1/2);
                for ib2 = 1:length(Pilares_glob)
                    if Pilares_glob(ib).ind == Pilares_glob(ib2).ind &&
dot(Pilares_glob(ib).dir,Pilares_glob(ib2).dir) == 0
                        Esf_Pilares_M3_Est(ib,:) =  Esf_Pilares_Est(ib2).M;
                        Esf_Pilares_V2_Est(ib,:) =  Esf_Pilares_Est(ib2).V;
                        Esf_Pilares_M3_Din(ib,:) =  Esf_Pilares_Din(ib2).M;
                        Esf_Pilares_V2_Din(ib,:) =  Esf_Pilares_Din(ib2).V;
                        break
                    end
                end
            end

            indb = 1;
            for ip = 1:length(obj.Porticos) %%%% Guardar os esf de volta nas barras
                obj.Porticos(ip).M3_V2_Est =
struct('M3',cell(obj.Porticos(ip).nPilares,1),'V2',0);
                obj.Porticos(ip).M3_V2_Din =
struct('M3',cell(obj.Porticos(ip).nPilares,1),'V2',0);
                for ib = 1 : obj.Porticos(ip).nPilares
                    obj.Porticos(ip).Esf_Estatica(ib).N = Esf_Pilares_N_Est(Pilares_glob(indb
+ ib - 1).ind,:);
                    obj.Porticos(ip).M3_V2_Est(ib).M3 = Esf_Pilares_M3_Est(indb + ib - 1,:);
                    obj.Porticos(ip).M3_V2_Est(ib).V2 = Esf_Pilares_V2_Est(indb + ib - 1,:);

                    obj.Porticos(ip).Comb_Final(ib).N = Esf_Pilares_N_Din(Pilares_glob(indb +
ib - 1).ind,:);
                    obj.Porticos(ip).M3_V2_Din(ib).M3 = Esf_Pilares_M3_Din(indb + ib - 1,:);
                    obj.Porticos(ip).M3_V2_Din(ib).V2 = Esf_Pilares_V2_Din(indb + ib -
1,:);%%%%%%%%%%
                end
                indb = indb + obj.Porticos(ip).nPilares;
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%    Verificações   %%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            for ip = 1:length(obj.Porticos)
                obj.Porticos(ip) = Verif(obj.Porticos(ip));
                if any(any(obj.Porticos(ip).Cond1))
                    aviso = ['O pórtico 2D nº ', num2str(ip),' não verifica os requisitos de
bom modelamento.'];
                    msgbox(aviso,'Aviso');
                end
                if any(any(obj.Porticos(ip).Cond2))
```

```
                    aviso = ['O pórtico 2D n°', num2str(ip),' não verifica a segurança em ',
num2str(sum(any(obj.Porticos(ip).Cond2,2))),' elementos.'];
                    msgbox(aviso,'Aviso');
                end
            end

            for ip = 1:length(obj.Porticos3D)
                obj.Porticos3D(ip) = Verif(obj.Porticos3D(ip));
                if any(any(obj.Porticos3D(ip).Cond1))
                    aviso = ['O pórtico 3D n° ', num2str(ip),' não verifica os requisitos de
bom modelamento.'];
                    msgbox(aviso,'Aviso');
                end
                if any(any(obj.Porticos3D(ip).Cond2))
                    aviso = ['O pórtico 3D n°', num2str(ip),' não verifica a segurança em ',
num2str(sum(any(obj.Porticos3D(ip).Cond2,2))),' elementos.'];
                    msgbox(aviso,'Aviso');
                end
            end

            for in = 1 : length(obj.Nucleos)
                obj.Nucleos(in) = Verif(obj.Nucleos(in));
%                 if any(any(obj.Porticos3D(ip).Cond1))
%                     aviso = ['O pórtico 3D n° ', num2str(ip),' não verifica os requisitos de
bom modelamento.'];
%                     msgbox(aviso,'Aviso');
%                 end
                if any(any(obj.Nucleos(in).Cond2))
                    aviso = ['O nucleo n°', num2str(in),' não verifica a segurança em ',
num2str(sum(any(obj.Nucleos(in).Cond2,2))),' elementos.'];
                    msgbox(aviso,'Aviso');
                end
            end

%             for ip = 1:length(obj.Porticos3D)
%                 obj.Porticos3D(ip) = inter(obj.Porticos3D(ip),obj.Miu_CQC);
%             end


            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%% Desenhos das Lajes %%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            fig = gcf;
            nfig = fig.Number;
            if ~isempty(fig.Name)
                nfig = nfig+1;
            end
            for im = Graficos.ModosPlanta %%%% Formatar as janelas para os graficos
                figure(nfig + floor((im - 1) / 3))
                set(gcf,'Position',[100, 200, 1200, 400]);
                set(nfig + floor((im - 1) / 3),'Name',['Modos ',
num2str(Graficos.ModosPlanta(1)), ' a ',
num2str(Graficos.ModosPlanta(end))],'NumberTitle','off')
            end
            xrect = [-.5, -.5, .5, .5, -.5];
            yrect = [-.5, .5, .5, -.5, -.5];
            for im = Graficos.ModosPlanta
                Modo = obj.Modos(:,im) * 1 / max(abs(obj.Modos(:,im))); %%% Normalizar
                figure(nfig + floor((im - 1) / 3))
                for il = 1:length(Lajes)
                    ip = find(AltPisos == Lajes(il).h) - 1;
                    coord = [xrect * Lajes(il).lx; yrect * Lajes(il).ly];
                    if abs(Modo(3 * ip)) > 10^-5
                        cir = cross([0; 0; Modo(3*ip)],[Modo(1 + 3 * (ip - 1)); Modo(2 + 3 *
(ip - 1)); 0]) * Modo(3 * ip) ^ (-2);
                        rot = [cos(Modo(3 * ip)), -sin(Modo(3 * ip));
                            sin(Modo(3 * ip)), cos(Modo(3 * ip))];
                        coord = rot * coord;
                        dist = rot * (Lajes(il).cg' - cir(1:2)); %%% vector distancia da laje
ao cir
                        coord(1,:) = coord(1,:) + cir(1) + dist(1);
                        coord(2,:) = coord(2,:) + cir(2) + dist(2);
                    else
                        coord(1,:) = coord(1,:) + Lajes(il).cg(1) + Modo(3 * ip - 2);
                        coord(2,:) = coord(2,:) + Lajes(il).cg(2) + Modo(3 * ip - 1);
                    end
                    subplot(1, 3, rem((im - 1), 3) + 1);
                    plot(coord(1,:),coord(2,:)); hold on;
                    daspect([1 1 1]);
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%      Animaçoes     %%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for im = 1:length(Graficos.AnimPlanta)
                %%%%% ANIMAÇÃO em planta

                fig = gcf;
                nfig = fig.Number;
                if ~isempty(fig.Name)
                    nfig = nfig+1;
                end

                Modo = obj.Modos(:,Graficos.AnimPlanta(im)) * 1 /
max(abs(obj.Modos(:,Graficos.AnimPlanta(im)))); %%% Normalizar
                figure(nfig); set(nfig,'Name',['Animação do Modo ',
num2str(Graficos.AnimPlanta(im))],'NumberTitle','off')
                cor = rand(length(AltPisos)-1,3);
                minx = Inf;
                maxx = -Inf;
                miny = Inf;
                maxy = -Inf;
                it = 0;
                for t = cos(linspace(0,5*pi,400))
                    it = it + 1;
                    for il = 1:length(Lajes)
                        ip = find(AltPisos == Lajes(il).h) - 1;
                        coord = [xrect * Lajes(il).lx; yrect * Lajes(il).ly];
                        if abs(Modo(3 * ip)) > 10^-5
```

```
                              Modo = Modo * max(5*abs(Modo(3:3:length(Modo)))) ^ -1;
                              cir = cross([0; 0; Modo(3*ip)],[Modo(1 + 3 * (ip - 1)); Modo(2 + 3
* (ip - 1)); 0]) * Modo(3 * ip) ^ (-2);
                              rot = [cos(Modo(3 * ip) * t), -sin(Modo(3 * ip) * t);
                                  sin(Modo(3 * ip) * t),  cos(Modo(3 * ip) * t)];
                              coord = rot * coord;
                              dist = rot * (Lajes(il).cg' - cir(1:2)); %%% vector distancia da
laje ao cir
                              coord(1,:) = coord(1,:) + cir(1) + dist(1);
                              coord(2,:) = coord(2,:) + cir(2) + dist(2);
                          else
                              coord(1,:) = coord(1,:) + Lajes(il).cg(1) + Modo(3 * ip - 2) * t;
                              coord(2,:) = coord(2,:) + Lajes(il).cg(2) + Modo(3 * ip - 1) * t;
                          end
                          if min(coord(1,:)) < minx
                              minx = min(coord(1,:));
                          end
                          if max(coord(1,:)) > maxx
                              maxx = max(coord(1,:));
                          end
                          if min(coord(2,:)) < miny
                              miny = min(coord(2,:));
                          end
                          if max(coord(2,:)) > maxy
                              maxy = max(coord(2,:));
                          end
                          if it > 80
                              fill(coord(1,:),coord(2,:),cor(ip,:),'EdgeColor','none'); hold on;
                              daspect([1 1 1]);
                          end
                      end
                      if it > 80
                          axis([minx, maxx, miny, maxy])
                          drawnow
                          hold off;
                      end
                  end
              end
              for im = 1:length(Graficos.Anim3D)
                  %%%%% ANIMAÇÃO 3D

                  fig = gcf;
                  nfig = fig.Number;
                  if ~isempty(fig.Name)
                      nfig = nfig+1;
                  end

                  Modo = obj.Modos(:,Graficos.Anim3D(im)) * 1 /
max(abs(obj.Modos(:,Graficos.Anim3D(im)))); %%% Normalizar
                  figure(nfig); set(nfig, 'Name',['Animação 3D do Modo ',
num2str(Graficos.Anim3D(im))],'NumberTitle','off')
                  minx = Inf;
                  maxx = -Inf;
                  miny = Inf;
                  maxy = -Inf;
```

```
                it = 0;
                for t = cos(linspace(0,5*pi,200))
                    it = it + 1;
                    for il = 1 : length(Lajes)
                        coord = zeros(3, 5 + 4 * 102);
                        ip = find(AltPisos == Lajes(il).h) - 1;
                        PeDir = AltPisos(ip + 1) - AltPisos(ip);
                        if abs(Modo(3 * ip)) > 10^-5
                            cir = cross([0; 0; Modo(3*ip)],[Modo(1 + 3 * (ip - 1)); Modo(2 + 3
* (ip - 1)); 0]) * Modo(3 * ip) ^ (-2);
                            rot = [cos(Modo(3 * ip) * t), -sin(Modo(3 * ip) * t), 0;
                                sin(Modo(3 * ip) * t),  cos(Modo(3 * ip) * t), 0;
                                0,                  0,                  1];
                            xlaje_0 = ([0, 1, 1, 0, 0] - .5) * Lajes(il).lx + Lajes(il).cg(1);
                            ylaje_0 = ([0, 0, 1, 1, 0] - .5) * Lajes(il).ly + Lajes(il).cg(2);
                            zlaje_0 = ones(1,5) * PeDir + (Lajes(il).h - PeDir);
                            coord(:,1:5) = rot * ([xlaje_0; ylaje_0; zlaje_0] - cir) + cir;
                            laj = coord(:,1:5);
                            coord_b_pilares = [xlaje_0(1:4); ylaje_0(1:4); zeros(1,4)];
                            if ip >= 2
                                cir_inf = cross([0; 0; Modo(3*(ip - 1))],[Modo(1 + 3 * (ip -
2)); Modo(2 + 3 * (ip - 2)); 0]) * Modo(3 * (ip - 1)) ^ (-2);
                                rot_inf = [cos(Modo(3 * (ip - 1)) * t), -sin(Modo(3 * (ip -
1)) * t), 0;
                                    sin(Modo(3 * (ip - 1)) * t),  cos(Modo(3 * (ip - 1)) * t),
0;
                                    0,                          0,                          1];
                                coord_b_pilares = rot_inf * (coord_b_pilares - cir_inf) +
cir_inf;
                            end
                        else
                            xlaje_0 = ([0, 1, 1, 0, 0] - .5) * Lajes(il).lx + Lajes(il).cg(1);
                            ylaje_0 = ([0, 0, 1, 1, 0] - .5) * Lajes(il).ly + Lajes(il).cg(2);
                            zlaje_0 = ones(1,5) * PeDir + (Lajes(il).h - PeDir);
                            coord(:,1:5) = [xlaje_0; ylaje_0; zlaje_0] + [Modo(1 + 3 * (ip -
1)); Modo(2 + 3 * (ip - 1)); 0] * t;
                            laj = coord(:,1:5);
                            coord_b_pilares = [xlaje_0(1:4); ylaje_0(1:4); zeros(1,4)];
                            if ip >= 2
                                coord_b_pilares = coord_b_pilares + [Modo(1 + 3 * (ip - 2));
Modo(2 + 3 * (ip - 2)); 0] * t;
                            end
                        end
                        pilares = repmat(0:.01:1,4,1);
                        x_pilares = diag(coord(1,1:4) - coord_b_pilares(1,1:4)) * (3 * pilares
.^ 2 - 2 * pilares .^ 3) + coord_b_pilares(1,1:4)';
                        y_pilares = diag(coord(2,1:4) - coord_b_pilares(2,1:4)) * (3 * pilares
.^ 2 - 2 * pilares .^ 3) + coord_b_pilares(2,1:4)';
                        z_pilares = pilares * PeDir + (Lajes(il).h - PeDir);
                        for i = 1:4
                            coord(:, (1:102) + 5 + 102 * (i - 1)) = [NaN(3,1),
[x_pilares(i,:); y_pilares(i,:); z_pilares(i,:)]];
                        end
                        if min(coord(1,:)) < minx
                            minx = min(coord(1,:));
```

```
                          end
                          if max(coord(1,:)) > maxx
                              maxx = max(coord(1,:));
                          end
                          if min(coord(2,:)) < miny
                              miny = min(coord(2,:));
                          end
                          if max(coord(2,:)) > maxy
                              maxy = max(coord(2,:));
                          end
                          if it > 80
%                                 plot3(coord(1,:),coord(2,:),coord(3,:)); hold on;
                              fill3(laj(1,:),laj(2,:),laj(3,:),'b'); hold on;
                              daspect([1 1 1]);
                          end
                      end
                      if it > 80
                          axis([minx, maxx, miny, maxy, 0, AltPisos(end)+1])
                          drawnow;
                          hold off;
                      end
                  end
              end

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              %%%%%%%%%%%%%%%%%%    Desenhos dos esforços    %%%%%%%%%%%%%%%%
              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

              if Graficos.Esf
                  fig = gcf;
                  nfig = fig.Number;
                  if ~isempty(fig.Name)
                      nfig = nfig + 1;
                  end
                  figure(nfig);
                  set(nfig,'Name','Esforcos ','NumberTitle','off'); %%%% Especificar o esforco

                  %%%% Desenho dos Porticos 2D
                  for ip = 1 : length(obj.Porticos)
                      Esf_Coloridos(obj.Porticos(ip));
                  end
                  %%%% Desenho dos Porticos 3D
                  for ip = 1 : length(obj.Porticos3D)
                      Esf_Coloridos(obj.Porticos3D(ip));
                  end
%                   %%%% Desenho dos Nucleos
                  for in = 1 : length(obj.Nucleos)
                      Esf_Coloridos(obj.Nucleos(in));
                  end
                  daspect([1 1 1]);
              end

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              %%%%%%%%%%%%%%%%%%   Desenhos das deformadas   %%%%%%%%%%%%%%%%
              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
            for im = Graficos.Def

                mult = 800; %% Multiplicador dos deslocamentos

                fig = gcf;
                nfig = fig.Number;
                if ~isempty(fig.Name)
                    nfig = nfig + 1;
                end
                figure(nfig);
                set(nfig,'Name',['Deformada do modo ', num2str(im)],'NumberTitle','off');

                if im == 0
                    qg = mult * q_est;
                else
                    qg = mult * obj.Modos(:,im);
                end

                %%%% Desenho dos Porticos 2D
                for ip = 1 : length(obj.Porticos)
                    Deformada(obj.Porticos(ip), qg);
                end
                %%%% Desenho dos Porticos 3D
                for ip = 1 : length(obj.Porticos3D)
                    Deformada(obj.Porticos3D(ip), qg);
                end
                %%%% Desenho dos Nucleos
                for in = 1:length(obj.Nucleos)
                    Deformada(obj.Nucleos(in), qg);
                end
                %%%% Desenho das Lajes
                xrect = [-.5, -.5, .5, .5];
                yrect = [-.5, .5, .5, -.5];
                for il = 1:length(Lajes)
                    ip = find(AltPisos == Lajes(il).h) - 1;
                    rot2 = [1, -(mult * obj.Modos(3 * ip, im));
                        (mult * obj.Modos(3 * ip, im)), 1];
                    coord2 = [xrect * Lajes(il).lx; yrect * Lajes(il).ly; ones(1,4) *
Lajes(il).h]; %%%%%%%[ 1, -teta; teta, 1]
                    coord2(1:2,:) = rot2 * (coord2(1:2,:) + Lajes(il).cg');
                    coord2(1,:) = coord2(1,:) + mult * obj.Modos(ip * 3 - 2,im); %%%%%%%%
                    coord2(2,:) = coord2(2,:) + mult * obj.Modos(ip * 3 - 1,im);
                    patch(coord2(1,:), coord2(2,:), coord2(3,:),'b'); hold on;

                end
                daspect([1 1 1]);
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%% Desenhos das partes condicionadas %%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            if Graficos.Cond
                fig = gcf;
                nfig = fig.Number;
                if ~isempty(fig.Name)
                    nfig = nfig + 1;
                end
                figure(nfig);
                set(nfig,'Name','Elementos Condicionados','NumberTitle','off');
                for ip = 1 : length(obj.Porticos)
                    ECond(obj.Porticos(ip));
                end
                for ip = 1 : length(obj.Porticos3D)
                    ECond(obj.Porticos3D(ip));
                end
                for in = 1 : length(obj.Nucleos)
                    ECond(obj.Nucleos(in));
                end
                daspect([1 1 1]);
            end
        end
    end
end
```

Appendix B – Codes for the tests performed

B.1) 3D Frame Model

```
Portico = [];

Portico3D = [];

Nucleo = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%% Dados Globais %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

AltPisos = 0:3:12;
Sismica = struct('Zona', 'A', 'Tipo', 1, 'ag_x_S', 2.5, 'q', 3,'amort',.05);

Sec(1) = struct('B', .4, 'H', .4, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6); %%%%
Pilares
Sec(2) = struct('B', .25, 'H', .4, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6); %%%%
Vigas

Lajes(1) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 1000 * 25 * 26, 'h', 3);
Lajes(2) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 1000 * 25 * 26, 'h', 6);
Lajes(3) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 1000 * 25 * 26, 'h', 9);
Lajes(4) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 1000 * 25 * 26, 'h', 12);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%% Porticos 3D %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% Pos_ref = [x, y, rot(graus)]; Fn = [no, fx, fy, fz, mx, my, mz]
clear Portico3D
Portico3D(1) = struct('vaos_x', [5, 5, 5, 5, 5], 'vaos_y', [5, 5, 6, 5, 5],...
    'Pisos', 1:5, 'SecPilares', 1, 'SecVigas', 2, 'Pos_ref', [-12.5; -13;
0],'Fn',[]);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%  Graficos  %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% 1-Modos em planta
%%%%% 2-Animaçao em planta
%%%%% 3-Configuraçao indeformada
%%%%% 4-Deformada
%%%%% 5-Elementos condicionados
%%%%% 6-Esforços

Graficos = struct('ModosPlanta', [], 'AnimPlanta', [], 'Anim3D', [], 'Indef',
false, 'Def', [], 'Cond', true, 'Esf', true);
```

B.2) 2D Frames modelling a regular 3D Frame

```
Portico = [];

Portico3D = [];

Nucleo = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%% Dados Globais %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

AltPisos = 0:3:12;
NumPisos = length(AltPisos); %%%
Sismica = struct('Zona', 'A', 'Tipo', 1, 'ag_x_S', 2.5, 'q', 3,'amort',.05);

Sec(1) = struct('B', .4, 'H', .4, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6); %%%%
Pilares
Sec(2) = struct('B', .25, 'H', .4, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6); %%%%
Vigas

Lajes(1) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 650000, 'h', 3);
Lajes(2) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 650000, 'h', 6);
Lajes(3) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 650000, 'h', 9);
Lajes(4) = struct('lx', 25, 'ly', 26, 'cg', [0, 0], 'M', 650000, 'h', 12);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%   Porticos   %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% Fn = [no, m, fh, fv], por linha
%%%% Pisos = indices dos pisos correspondentes as alturas em AltPisos, pode
%%%% ser um cell para ter alturas diferentes; uma linha por vao.
clear Portico

Portico(1) = struct('dir', [1, 0], 'origem', [-12.5, -13], 'vaos', [5, 5, 5, 5,
5],...
    'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 2.5, 'Fn', []);

Portico(2) = struct('dir', [1, 0], 'origem', [-12.5, -8], 'vaos', [5, 5, 5, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 5, 'Fn', []);
Portico(3) = struct('dir', [1, 0], 'origem', [-12.5, -3], 'vaos', [5, 5, 5, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 5.5, 'Fn', []);
Portico(4) = struct('dir', [1, 0], 'origem', [-12.5, 3], 'vaos', [5, 5, 5, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 5.5, 'Fn', []);
Portico(5) = struct('dir', [1, 0], 'origem', [-12.5, 8], 'vaos', [5, 5, 5, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 5, 'Fn', []);
Portico(6) = struct('dir', [1, 0], 'origem', [-12.5, 13], 'vaos', [5, 5, 5, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 2.5, 'Fn', []);
Portico(7) = struct('dir', [0, 1], 'origem', [-12.5, -13], 'vaos', [5, 5, 6, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 2.5, 'Fn', []);
Portico(8) = struct('dir', [0, 1], 'origem', [-7.5, -13], 'vaos', [5, 5, 6, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 5, 'Fn', []);
```

```
Portico(9) = struct('dir', [0, 1], 'origem', [-2.5, -13], 'vaos', [5, 5, 6, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 5, 'Fn', []);
Portico(10) = struct('dir', [0, 1], 'origem', [2.5, -13], 'vaos', [5, 5, 6, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 5, 'Fn', []);
Portico(11) = struct('dir', [0, 1], 'origem', [7.5, -13], 'vaos', [5, 5, 6, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 5, 'Fn', []);
Portico(12) = struct('dir', [0, 1], 'origem', [12.5, -13], 'vaos', [5, 5, 6, 5, 5],
'Pisos', 5, 'SecPilares', 1, 'SecVigas', 2, 'L_inf', 2.5, 'Fn', []);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%  Graficos  %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% 1-Modos em planta
%%%%% 2-Animaçao em planta
%%%%% 3-Configuraçao indeformada
%%%%% 4-Deformada
%%%%% 5-Elementos condicionados
%%%%% 6-Esforços

Graficos = struct('ModosPlanta', [], 'AnimPlanta', [], 'Anim3D', [], 'Indef',
false, 'Def', [], 'Cond', true, 'Esf', true);
```

B.3) Irregular Building

```
Portico = [];

Portico3D = [];

Nucleo = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%% Dados Globais %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

AltPisos = 0:3:12;
NumPisos = length(AltPisos); %%%
Sismica = struct('Zona', 'A', 'Tipo', 1, 'ag_x_S', 2.5, 'q', 3,'amort',.05);

Sec(1) = struct('B', .4, 'H', .4, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6); %%%%
Pilares
Sec(2) = struct('B', .3, 'H', .6, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6); %%%% Vigas
Sec(3) = struct('B', .4, 'H', .4, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6);
Sec(4) = struct('B', .4, 'H', .4, 'E', 30 * 10 ^ 9, 'fcd', 30 * 10 ^ 6); %%%Pilares

Lajes(1) = struct('lx', 15, 'ly', 18, 'cg', [7.5, 9], 'M', 1000 * 15 * 18, 'h', 3);
Lajes(2) = struct('lx', 22.5, 'ly', 12, 'cg', [11.25, 24], 'M', 1000 * 22.5 * 12,
'h', 3);
Lajes(3) = struct('lx', 15, 'ly', 18, 'cg', [7.5, 9], 'M', 1000 * 15 * 18, 'h', 6);
Lajes(4) = struct('lx', 22.5, 'ly', 12, 'cg', [11.25, 24], 'M', 1000 * 22.5 * 12,
'h', 6);
Lajes(5) = struct('lx', 15, 'ly', 18, 'cg', [7.5, 9], 'M', 1000 * 15 * 18, 'h', 9);
Lajes(6) = struct('lx', 15, 'ly', 18, 'cg', [7.5, 9], 'M', 1000 * 15 * 18, 'h',
12);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%   Porticos   %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% Fn = [no, m, fh, fv], por linha
%%%% Pisos = indices dos pisos correspondentes as alturas em AltPisos, pode
%%%% ser um cell para ter alturas diferentes; uma linha por vao.
clear Portico

Portico(1) = struct('dir', [1, 0], 'origem', [0, 0], 'vaos', [7.5, 7.5], 'Pisos',
5, 'SecPilares', 3, 'SecVigas', 2, 'L_inf', 3, 'Fn', []);
Portico(2) = struct('dir', [1, 0], 'origem', [0, 18], 'vaos', [7.5, 7.5, 7.5],
'Pisos', [5, 5, 3], 'SecPilares', [4, 1, 4, 4], 'SecVigas', 2, 'L_inf', 3, 'Fn',
[]);
Portico(3) = struct('dir', [1, 0], 'origem', [0, 24], 'vaos', [7.5, 7.5, 7.5],
'Pisos', 3, 'SecPilares', [4, 1, 1, 4], 'SecVigas', 2, 'L_inf', 6, 'Fn', []);
Portico(4) = struct('dir', [1, 0], 'origem', [0, 30], 'vaos', [7.5, 7.5, 7.5],
'Pisos', 3, 'SecPilares', [3, 3, 3, 4], 'SecVigas', 2, 'L_inf', 3, 'Fn', []);
Portico(5) = struct('dir', [0, 1], 'origem', [0, 0], 'vaos', [6, 6, 6, 6, 6],
'Pisos', [5, 5, 5, 3, 3], 'SecPilares', [4, 3, 3, 3, 3, 4], 'SecVigas', 2, 'L_inf',
3.75, 'Fn', []);
```

```
Portico(6) = struct('dir', [0, 1], 'origem', [7.5, 18], 'vaos', [6, 6], 'Pisos',
[3, 3], 'SecPilares', [1, 1, 4], 'SecVigas', 2, 'L_inf', 7.5, 'Fn', []);
Portico(7) = struct('dir', [0, 1], 'origem', [15, 0], 'vaos', [6, 6, 6, 6, 6],
'Pisos', [5, 5, 5, 3, 3], 'SecPilares', [4, 3, 3, 3, 1, 4], 'SecVigas', 2, 'L_inf',
4, 'Fn', []);
Portico(8) = struct('dir', [0, 1], 'origem', [22.5, 18], 'vaos', [6, 6], 'Pisos',
[3, 3], 'SecPilares', [3, 3, 3], 'SecVigas', 2, 'L_inf', 3.75, 'Fn', []);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%   Nucleos   %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  ->  Nucleo(1) = struct('Nos', [x1, y1; ...], 'Paredes', [n1, n2; ...], 't', t ou
[t1, ..., tn], 'Pisos', [p1, ...], 'E', E);
%%%%%%%%         Nos = coordenadas em planta, 1 par de coord por linha;
%%%%%%%%         t = espessura, pode ser 1 so valor ou um vector de tamanho igual
ao n de paredes (incluindo paredes ligadas);
%%%%%%%%         Pisos = Indices do vector AltPisos que correspondem aos pisos que
o nucleo toca
%%%%%%%%         E = Modulo de elasticidade em GPa
%%%%%%%%         ParedesLigadas = [Parede,%Parede,%] por linha
clear Nucleo

Nucleo(1) = struct('Nos', [10.5, 6; 4.5, 6; 4.5, 12; 10.5, 12], 'Paredes', [1, 2;
2, 3; 3, 4],...
    'ParedesLigadas', [], 't', .15, 'Pisos', 1:5, 'E', 30 * 10 ^ 9,...
    'fcd', 30 * 10 ^ 6, 'A_inf', [6, 3.75]);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%  Graficos  %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% 1-Modos em planta
%%%%% 2-Animaçao em planta
%%%%% 3-Configuraçao indeformada
%%%%% 4-Deformada
%%%%% 5-Elementos condicionados
%%%%% 6-Esforços

Graficos = struct('ModosPlanta', [], 'AnimPlanta', [1:3], 'Anim3D', [], 'Indef',
true, 'Def', [3], 'Cond', true, 'Esf', true);
```