Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Modeling and Simulation of 3D Frame Structures

Emanuel Antônio Parreiras

JUIZ DE FORA
JULHO, 2019

Modeling and Simulation of 3D Frame Structures

Emanuel Antônio Parreiras

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Marcelo Bernardes Vieira
Coorientador: Marcelo Miranda Barros

JUIZ DE FORA
JULHO, 2019

Modeling and Simulation of 3D Frame Structures

Emanuel Antônio Parreiras

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Marcelo Bernardes Vieira
Doutor em Ciência da Computação

Marcelo Miranda Barros
Doutor em Modelagem Computacional

Marcelo Caniato Renhe
Doutor em Engenharia de Sistemas e Computação

George Oliveira Ainsworth Junior
Doutor em Engenharia Civil

JUIZ DE FORA
16 DE JULHO, 2019

Resumo

As estruturas em frame são estruturas combinadas de vigas, pilares e lajes, feitas para resistir às cargas e à gravidade. Uma grande parte de estruturas modeladas por um engenheiro civil podem ser visualizadas como um frame 2D ou 3D. Neste trabalho vamos ter uma visão introdutória de como é feita uma análise estrutural linear em 3D, passando pelos sistemas de molas, análise sistemas de elementos em 2D e 3D com a aplicação de matriz de transformação. Passamos também pelos métodos numéricos para solução de sistemas lineares, formas de armazenamento em memória principal para a otimização de operações sobre o modelo virtual. Utilizamos os dados da literatura para construir um protótipo funcional para análise estrutural linear de estruturas em frame, por fim, compararemos os resultados obtidos com softwares comerciais.

Palavras-chave: Análise Estrutural, Estruturas em Frame, Método da Rigidez, Métodos Numéricos, Computação Gráfica, Modelagem em Tempo Real.

Abstract

The frame structures are combined structures of beams, columns and slabs, made to resist to loads and gravity. The majority of structures modeled by a civil engineer can be viewed as a frame 2D or 3D. In this work we have an introductory view of how a 3D linear structural analysis is made, passing through the spring systems, analyzing 2D and 3D element systems using the transformation matrix. We also discuss the numerical methods for solution of linear systems, forms of storage in main memory for the optimization of operations on the virtual model. We use the literature data to construct a functional prototype for linear frame analysis, and finally, we will compare the results obtained with commercial softwares.

Keywords:Structural Analysis, Frame Structures, Stiffness Method, Numerical Methods, Computer Graphics, Real Time Modeling

Agradecimentos

Primeiramente gostaria de agradecer aos meus orientadores Marcelo Bernardes e Marcelo Miranda por toda a paciência e ensinamentos que foram cruciais para a realização desse projeto. A Adriele Valle, já que sem toda a sua ajuda e visão esse trabalho não teria saído do papel.

Aos meus pais, Regina e Manoel, que se esforçaram ao máximo para me dar uma educação de qualidade.

À minha namorada Lilian Amorim que em todos esses anos juntos me deu forças para superar as dificuldades da vida.

E a todos os professores e amigos que contribuíram de alguma forma para o desenvolvimento desse projeto.

Contents

List of Figures

List of Tables

Abbreviation List

DOF	Degrees of Freedom
CPU	Central Processing Unit
GPU	Graphics Processing Unit
PUC-Rio	Pontíficia Universidade Católica do Rio de Janeiro
2D	Two dimensions
3D	Three dimensions
S.I.	International System of Units
CSR	Compressed Sparse Row
CSC	Compressed Sparse Column
MSE	Mean Squared Error
CAD	Computer-Aided Design
GUI	Graphics User Interface

1 Introduction

Softwares for structural analysis are present in the life of all civil engineers and architects, be during the graduation or out of university. But most of the tools available today are not have free access, which makes it difficult for the usage out of the classroom. An alternative is the Ftool (MARTHA, 2019), developed from 1998, it is one of the most famous tools of structural analysis of 2D frames in Brazil. But the Ftool has limitations, such as all free software, it makes only the analysis of plane frames, not computing structure dynamic analysis, etc. The analyses of structures modeled in 3D frames are possible using plane frames to simulate this structure type but are necessary to make simplifications in the model to get an approximation. The commercial software provides a range of analysis possibilities that we did not find in free software, but the license cost is prohibitive.

1.1 Problem Definition

Frame structures are constructions composed of parts or elements linked together made to support and resist loads. Usually, a frame structure is represented by a combination of beams, columns, and slabs to resist gravity force and wind loads. For static analysis, the structure needs to have all possible movements restricted by supports. When subject to loads a structure undergoes displacements and for small enough displacements, the relations between loads and displacements are linear. One basic hypothesis to perform a linear analysis is that the configuration of the structure post deformation is very close to the original state, that is, the displacements are small compared to the dimensions of the structure. Thus, cause (loads) and effect (displacements) relates linearly, modeled by a linear system of equations.

1.2 Objectives

This work aims to create a functional prototype of structural analysis that enables the user to create virtual models of 3D frames that be analyzed. In course of this work, we describe how to build a software that analyzes 3D frames, we create a prototype that is used to validate the mathematical method and the numerical method, and in the end, have a software that allows future extensions.

2 Mathematical Modeling of Frame Structures

The problem of solving a structure may be posed by the equilibrium equations derived from Newton's movement law for statics and Hooke's law. If the displacements of a structure, i.e. its degrees of freedom, are represented by the displacements of the extremities of the bar, defined as nodes, the law of statics needs to be satisfied for all degrees of freedom. Depending on the characteristic of the bar element of a model, e.g. beam or 3D frame, there are different quantities of degrees-of-freedom (DOF) per node. Assuming that elements behave as linearly elastic solids then the law of equilibrium for the whole structure resumes to the relation $F = KU$ where F is the force vector, K is the stiffness matrix and U is the displacement vector. The order of the linear system is given by the product between the number of nodes and the number of DOF per node. The linear system is singular unless we pose adequate restrictions to the movement, represented by a set of null or known displacements/rotations, called kinematic boundary conditions.

To show the methodology for more general frame structures we present a sequence of models, namely spring, 2D frame, and 3D frame, with increasing complexity and representativeness. A spring element has one degree-of-freedom per node, whereas a 2D frame has three and a 3D frame has six DOF per node. To model a whole elastic structure, the stiffnesses of all elements are assembled into a global stiffness matrix where the stiffness of each element is known or deduced a priori, for a given type of element. This method is called the stiffness method.

2.1 Model 1: System of Springs

A spring is the most simple model for an elastic structure, described by only two (DOF), one per node, illustrated by Fig. 2.1.

2.1.1 Stiffness Matrix of a Spring

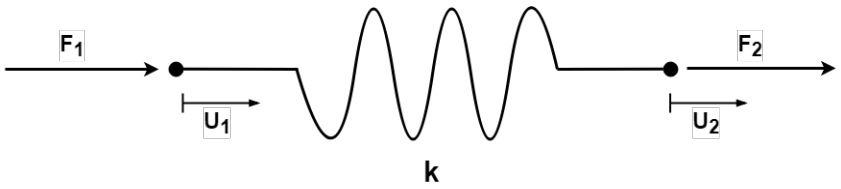


Figure 2.1: A spring with stiffness k and its displacements and forces.

Considering the spring in equilibrium by Newton's law $\sum F = 0$ we obtain:

$$\begin{aligned} F_1 + F_{r_1} &= 0, \\ F_2 + F_{r_2} &= 0. \end{aligned} \tag{2.1}$$

where F_1 and F_2 are external forces whereas F_{r_1} and F_{r_2} are the internal forces due to the spring restoring forces. Assuming that the spring behaves linearly, i.e. follows Hooke's law ($F_r = -k\Delta u$), we have that $F_{r_1} = -k(u_1 - u_2)$ and $F_{r_2} = -k(u_2 - u_1)$ where the minus sign indicates that the force of the spring is contrary to the relative displacement, i.e. depending if the spring is elongated or stretched. Therefore the equilibrium may be written as the following matrix Eq. 2.2.

$$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \tag{2.2}$$

or in compact form $[F] = [K][U]$, where $[F]$ is the vector of external forces, $[K]$ is the stiffness matrix of a spring and $[U]$ is the displacement vector.

2.1.2 A System of Springs

The behavior of one spring is established from Eq. 2.2, thus, it is possible to build systems with interconnected springs. The most simple example is a system composed of two springs with stiffnesses k_1 and k_2 in series, presenting three DOF as illustrates in Fig. 2.2.

Using Newton's third law, we can write Eq. 2.3 in matrix form.

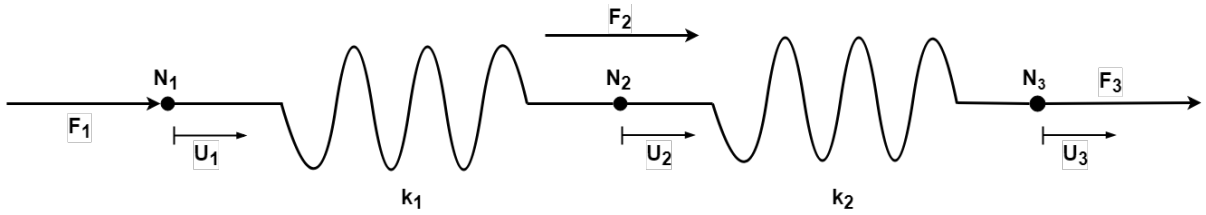


Figure 2.2: System with two springs with stifnesses k_1 and k_2, and their displacements and forces.

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}, \tag{2.3}$$

we notice that the global stiffness matrix is a superposition of two elementary spring stiffness matrix. Note that the stiffness associated with the degree of freedom u_2 is the sum of k_1 and k_2 since to displace it needs to deform both springs. System 2.3 may be written in compact form as Eq. 2.4.

$$[F] = [K_G][U], \tag{2.4}$$

where the $[K_G]$ is the global stiffness matrix of the system.

If we try to solve the previous system, we fing a problem, $det[K_G] = 0$, this implies that K_G is singular. This means that the system is not in equilibrium since there is no restriction to the movement. Then it is necessary to establish a given suitable set of restrictions to attain equilibrium. The restrictions are imposed as known displacements which usually are null. Thus we solve the subsystem associated with the unknown displacements and known forces. After knowing the displacements we solve the system to find the unknown forces.

Example 1

We build a subsystem to analyze the boundary conditions for the matrix Eq. 2.3, by adding a movement restriction to the node N_3, therefore $u_3 = 0$.

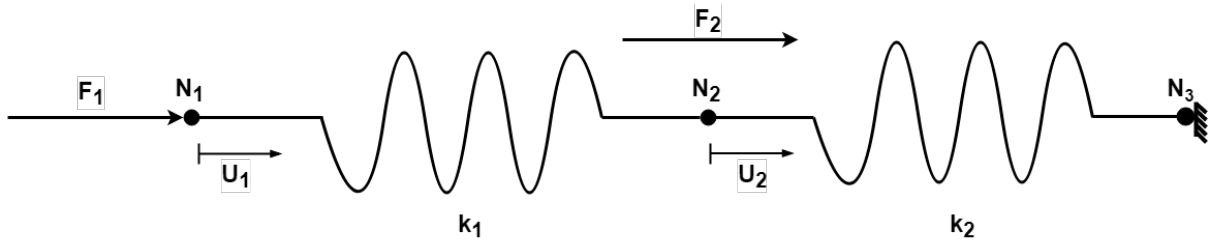


Figure 2.3: System of two springs and one movement restriction.

Mounting the subsystem based in Eq. 2.3 with this restriction, we obtain Eq. 2.5:

$$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 + k_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \tag{2.5}$$

and solving the subsystem we find the displacements:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \frac{F_1(k_1+k_2)+F_2k_1}{(k_1k_2)} \\ \frac{F_1+F_2}{k_2} \\ 0 \end{bmatrix}. \tag{2.6}$$

Creating a Subsystem with Two Movement Restrictions

To finish the analysis of the example with two springs, we perform two-movement restrictions, one in node N_1 and another in N_3. In this case we have null displacements at nodes, $u_1 = 0$ and $u_3 = 0$, consequently the external forces $\vec{F_1}$ and $\vec{F_3}$ have no influence in finding the displacement.

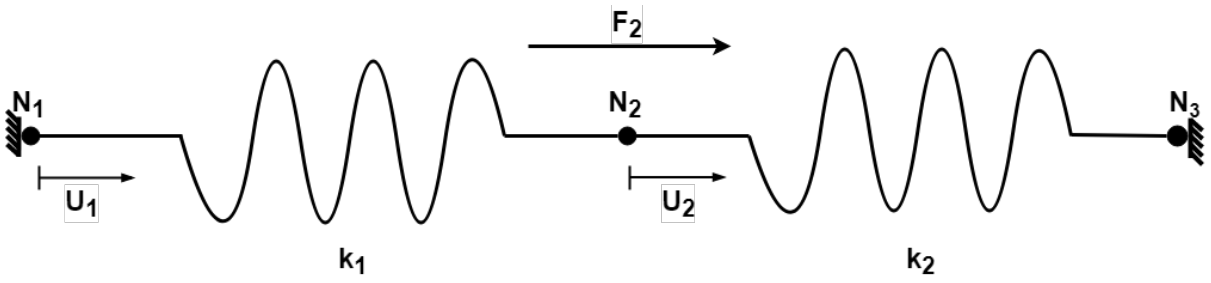


Figure 2.4: System of springs with two movement restrictions.

Mounting the subsystem based in the Eq. 2.3 with two restrictions, we obtain the Eq. 2.7.

$$\left[F_2\right] = \left[k_1 + k_2\right]\left[u_2\right], \tag{2.7}$$

and solving the subsystem we find the displacements presented in Eq. 2.6.

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{F_2}{k_1+k_2} \\ 0 \end{bmatrix}. \tag{2.8}$$

Analyzing the result in the Eq. 2.8 obtained from Eq. 2.7, u_2 depend directly of the direction of $\vec{F_2}$. We can verify that Eq. 2.7 is equivalent to a spring association in parallel.

2.2 Model 2: 2D Framed Systems

A 2D frame element posses three DOF per node, namely horizontal displacement, vertical displacement and rotation. Then each element posses six DOF as illustrates in Fig. 2.5.

Figure 2.5: An element with three DOF per node and the forces exerted in each node.

We can consider a plane structure, so we need to rewrite the equilibrium Eq. 2.1 for this model, including more two DOF, we obtain the Eqs. 2.9.

$$\sum F_x = 0, \quad \sum F_y = 0, \quad \sum M_z = 0, \tag{2.9}$$

for each node, when $\sum M_z$ is the sum of all moments that produce rotation in an additional

axis z perpendicular to the xy plane. For each external forces exerted in some node of the system, we can decompose it in three components, two axial forces and one moment.

The angles θ_1 and θ_2 are the angles produced by the moments M_1 and M_2 respectively. Assuming $C(x)$ as the curve that represent the deformed element, we can calculate the tangent straights t_1 and t_2 from the points of $C(0)$ and $C(L)$. The calculated angles are the angles between the original element axis s and the tangent lines t_1 and t_2 as illustrated by Fig. 2.6.

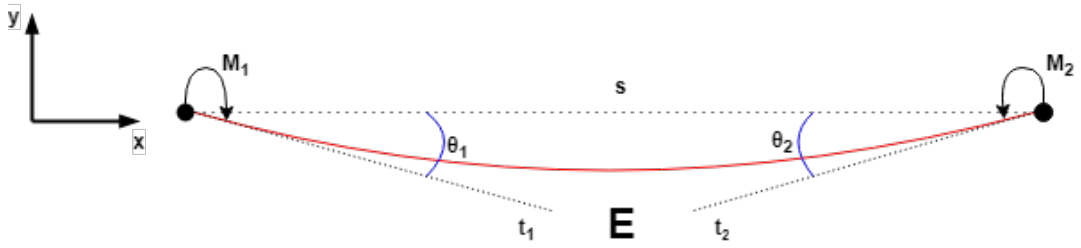


Figure 2.6: A 2D element with two moments, M_1 and M_2, exerted on their nodes.

Using the Eq. 2.9, and using the third Newton's law, we can separate F_x, F_y and M_z into action forces and restoring forces on each node. Thus we can define the resulting action forces as H, V, M and the resulting restoring forces as H_r, V_r, M_r, and finally, we obtain the equilibrium Eq. 2.10.

$$\begin{aligned} H_1 + H_{r_1} = 0, \quad V_1 + V_{r_1} = 0, \quad M_1 + M_{r_1} = 0, \\ H_2 + H_{r_2} = 0, \quad V_2 + V_{r_2} = 0, \quad M_2 + M_{r_2} = 0. \end{aligned} \tag{2.10}$$

The nodal moments M_1 and M_2 produce the nodal rotations θ_1 and θ_2, according to McCormac (2009) these moments can be expressed by Eq. 2.11.

$$\begin{aligned} M_1 &= \left(\frac{4EI}{L}\right)\theta_1 + \left(\frac{2EI}{L}\right)\theta_2, \\ M_2 &= \left(\frac{2EI}{L}\right)\theta_1 + \left(\frac{4EI}{L}\right)\theta_2, \end{aligned} \tag{2.11}$$

where I is the element moment of inertia, A is the section area and L is the length of the element. The Young modulus E is a material property that measures the difficulty in deforming a solid element composed by this material. E can be calculated using the Eq. 2.12

$$E = \frac{FL_0}{A\Delta L}, \tag{2.12}$$

where F, A, ΔL and L_0 are illustrated by Fig. 2.7. In SI base, the E unit is Pascal(Pa).

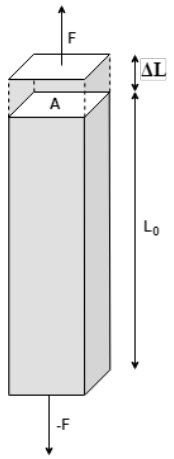


Figure 2.7: A solid deformed by a force F.

However, the properties as A, I, and L can be calculated from geometrical form of the element. The moment of inertia of a section is calculated using a perpendicular axis z, where the rotation axis is parallel to the z and passes through the section center. McCormac (2009) defines the horizontal, and the vertical displacements as the Eqs. 2.13.

$$\begin{aligned}
H_1 &= \left(\frac{EA}{L}\right) h_1 - \left(\frac{EA}{L}\right) h_2, \\
V_1 &= \left(\frac{12EI}{L^3}\right) v_1 + \left(-\frac{12EI}{L^3}\right) v_2, \\
H_2 &= -\left(\frac{EA}{L}\right) h_1 + \left(\frac{EA}{L}\right) h_2, \\
V_2 &= \left(-\frac{12EI}{L^3}\right) v_1 + \left(\frac{12EI}{L^3}\right) v_2.
\end{aligned} \tag{2.13}$$

Finally, for an isolated element, we obtain Eq. 2.14.

$$\begin{bmatrix} H_1 \\ V_1 \\ M_1 \\ H_2 \\ V_2 \\ M_2 \end{bmatrix} = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \begin{bmatrix} h_1 \\ v_1 \\ \theta_1 \\ h_2 \\ v_2 \\ \theta_2 \end{bmatrix}, \tag{2.14}$$

where E is the Young modulus, A is the section area, I is the section moment of inertia and L is the length of the element.

2.2.1 A 2D frame

The next step with the 2D analysis using the displacement method is to create a model with more elements. The assembly of a 2D model is not more complicated than a simple system of springs that we made, but in this case, we need to use the artifice of transforming the element local coordinates into global coordinates during the analysis. The example that we going to use is illustrated in Fig. 2.8.

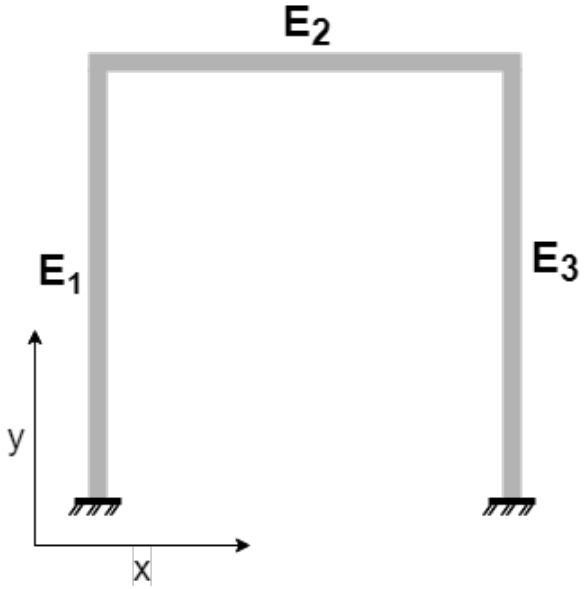


Figure 2.8: A plane frame example.

To turn the example of Fig. 2.8 in an element-node model, we will use the

concepts of the isolated element. First, we need to decompose the structure in three elements, E_1, E_2 and E_3, where each element will have its own properties of geometry and materials.

Using the stiffness method to calculate the displacements of each node, we need to analyze each element separately, lastly, we create a new diagram to represent the model, illustrated by Fig. 2.9.

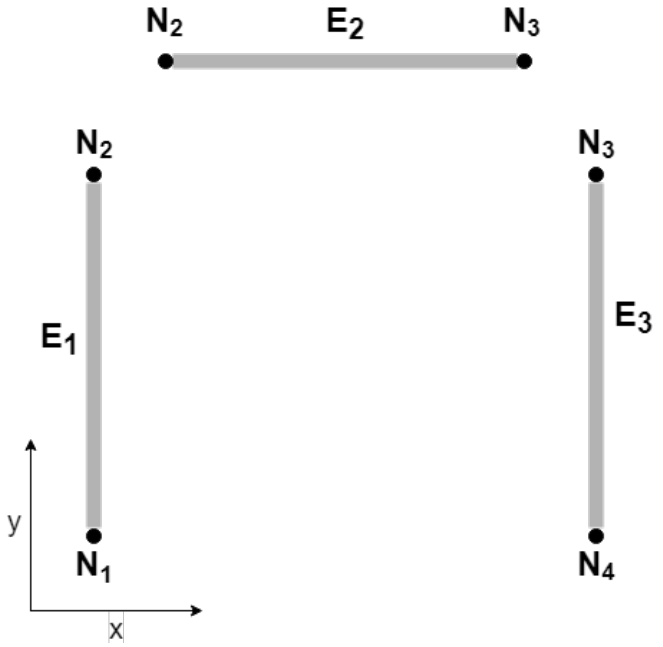


Figure 2.9: A plane frame decomposed in an element-node model.

First, to obtain the system stiffness matrix, we need to analyze locally each element. But the matrix Eq. 2.14 was made analyzing the element over the x axis, thus, we need to find a way to generalize the matrix equation. If we observe the example illustrated in Fig. 2.9, one can note that the E_1 and E_3 elements have components in the y axis. In the literature, this problem is treated using a transformation matrix, that is used to transform the stiffness matrix from element local coordinates to system global coordinates. According to McCormac (2009), we can define the rotation matrix Eq. 2.15.

$$[R] = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.15}$$

where ϕ is the angle between the direction cosine and x-axis. The direction cosine $\vec{C}$ can

be defined as:

$$\vec{C} = \frac{\vec{V}}{\left\|\vec{V}\right\|}, \quad \vec{V} = \vec{N_j} - \vec{N_i},$$

where $\vec{V}$ is the direction vector, $\vec{N_i}$ and $\vec{N_j}$ are the coordinates of the nodes that define an element. The $cos(\phi)$ and $sin(\phi)$ can be calculated using $\vec{C}$ as:

$$cos(\phi) = \frac{Cx}{\left\|\vec{C}\right\|}, \quad sin(\phi) = \frac{Cy}{\left\|\vec{C}\right\|}.$$

The transformation matrix is composed by a linear combination of matrix Eq. 2.15. According to McCormac (2009), we can define the transformation matrix Eq. 2.16.

$$[T] = \begin{bmatrix} \cos\phi & \sin\phi & 0 & 0 & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\phi & \sin\phi & 0 \\ 0 & 0 & 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.16}$$

Using the transformation matrix $[T]$ and the local stiffness matrix K^* obtained from matrix Eq. 2.14, we will use the Eq. 2.17 to convert the element local coordinates to system global coordinates.

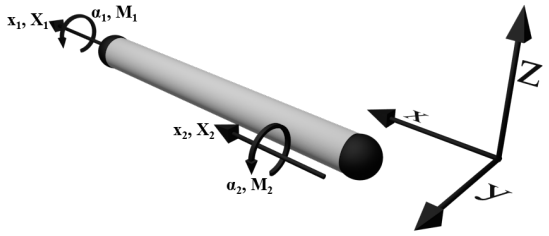
$$[F] = [T]^t[K^*][T][U], \tag{2.17}$$

where $[F]$ are the external forces in global coordinates exerted on the element nodes, $[U]$ are the displacements/rotations caused by $[F]$.

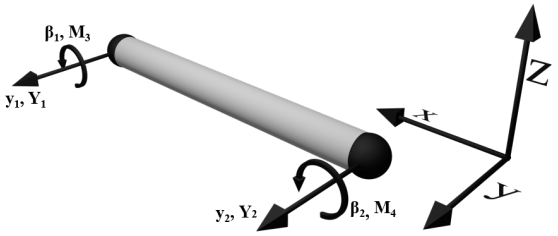
The assembly of the linear system with stiffness matrices of all elements obeys the principle of superposition of forces, where the final stiffness matrix is the sum of all stiffness matrices of each separate element.

2.3 Model 3: 3D Framed Systems

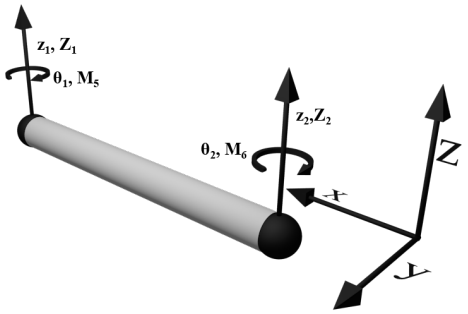
The three-dimensional case is an extension from the two-dimensional case that we discussed in Section 2.2, where it is necessary to define more DOF. Since we have an extra axis on the system z, there be three more DOF per node. In the example illustrated in Fig. 2.10, we have six DOF per node, where three are displacements and three are rotations.



(a) A 3D element and its displacements and rotations on x-axis.



(b) A 3D element and its displacements and rotations on y-axis.



(c) A 3D element and its displacements and rotations on the z-axis.

Figure 2.10: An element with six DOF per node.

We preserve the equilibrium Eqs. 2.9, however, we rewrite it including more three DOF per node in Eq. 2.18.

$$\begin{aligned} \sum F_x = 0, \quad \sum F_y \quad = 0, \quad \sum F_z = 0, \\ \sum M_x = 0, \quad \sum M_y \quad = 0, \quad \sum M_z = 0. \end{aligned} \tag{2.18}$$

For each force exerted on the nodes of an element, we can decompose the equilibrium equations in Eq. 2.19, where X, Y, Z are the resulting action forces, X_r, Y_r and Z_r are the resulting restoring forces, M is the resulting moment exerted in a DOF of rotation and finally M_r is the resulting restoring moment. The indices can be observed in Fig. 2.10.

$$\begin{aligned} X_1 + X_{r_1} = 0, \quad M_1 + M_{r_1} = 0, \\ X_2 + X_{r_2} = 0, \quad M_2 + M_{r_2} = 0, \\ Y_1 + Y_{r_1} = 0, \quad M_3 + M_{r_3} = 0, \\ Y_2 + Y_{r_2} = 0, \quad M_4 + M_{r_4} = 0, \\ Z_1 + Z_{r_1} = 0, \quad M_5 + M_{r_5} = 0, \\ Z_2 + Z_{r_2} = 0, \quad M_6 + M_{r_6} = 0. \end{aligned} \tag{2.19}$$

Extending the Eq. 2.11 and Eq. 2.13, adding more three DOF, one for translation, and two for rotations, we can write the z-axis displacement and the moments, according to Kattan (2007), on x-axis and y-axis in the Eq. 2.20.

$$\begin{aligned} Z_1 &= \frac{12EI_z}{L^3}(z_1 - z_2), \\ Z_2 &= \frac{12EI_z}{L^3}(z_2 - z_1), \\ M_1 &= \frac{GJ}{L}(\alpha_1 - \alpha_2), \\ M_2 &= \frac{GJ}{L}(\alpha_2 - \alpha_1), \\ M_3 &= \frac{4EI_y}{L^2}\beta_3 + \frac{2EI_y}{L}\beta_4, \\ M_4 &= \frac{2EI_y}{L^2}\beta_3 + \frac{4EI_y}{L}\beta_4, \end{aligned} \tag{2.20}$$

where $J = I_y + I_z$ according to Rethwisch (2007). The shear modulus G is a material property defined as the ratio of shear stress to the shear strain and can be calculated using the Eq. 2.21.

$$G = \frac{F/A}{\Delta x/l}, \tag{2.21}$$

where F, A, Δx and l are illustrated by Fig. 2.11. In SI base, the G unit is Pascal(Pa).

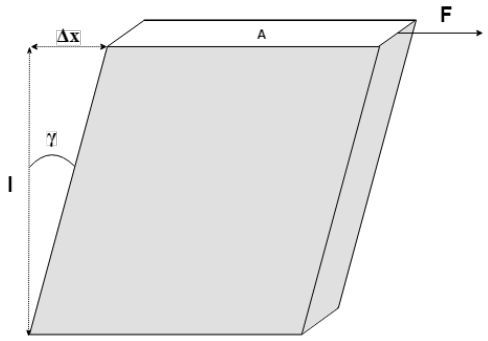


Figure 2.11: A solid deformed by a force F.

Usually, the shear modulus G is not defined directly on software, instead it is defined, for an isotropic material, by the Poisson coefficient ν, that can be written in the form:

$$\nu = \left(\frac{E}{2G}\right) - 1.$$

Using the equilibrium Eq. 2.20, according to Weaver (1980), we can define the stiffness matrix for a frame element with six DOF per node as Eq. 2.22.

$$[K^*] = \begin{bmatrix}
\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} \\
0 & 0 & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 \\
0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 \\
0 & 0 & -\frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 \\
0 & \frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} \\
-\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} & 0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} \\
0 & 0 & -\frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 & 0 & 0 & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 \\
0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 \\
0 & 0 & -\frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & 0 \\
0 & \frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L}
\end{bmatrix}. \quad (2.22)$$

2.3.1 3D Rotation Matrices

To generalize the stiffness matrix K^*, we need to use a transformation matrix. At 3D case, is more complex to visualize the transformation from rotation matrices, for a space transformation we need three rotation matrices, one for each axis.

Z-Rotation

The rotation about z-axis is equivalent of the rotation described in Section 2.2.1 for 2D case. Using the direction cosine of an element in $\mathbb{R}^3$, to rotate around an axis, we fix one, z-axis, and we rotate the direction cosine on a perpendicular plane, e.g. xy-plane, as illustrated in Fig. 2.12. Note that whole the analysis, we placed the element at x -axis, and the function of the rotation matrices is to place the element in the space position.

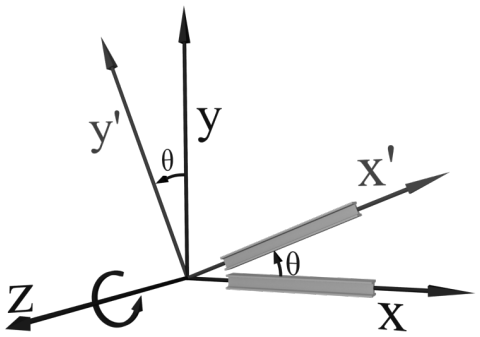


Figure 2.12: Rotation around z-axis.

In the rotation illustrated by Fig. 2.12, we transform the original element axis x, y, z to an alternative element axis x', y', z. According to Beaufait (1971), we can use the rotation matrix Eq. 2.23.

$$[R_z] = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.23}$$

where θ is the angle around z-axis and can be calculated using the direction cosine as in Eq. 2.24.

$$\begin{aligned} C_{xz} &= \|proj_{xz}C\|, \\ sin(\theta) &= C_{xz}, \\ cos(\theta) &= C_y. \end{aligned} \tag{2.24}$$

Y-Rotation

Similar with the Z-Rotation, we fix the y-axis, and we rotate the element on the perpendicular plane, zx-plane, as illustrated by Fig. 2.13.

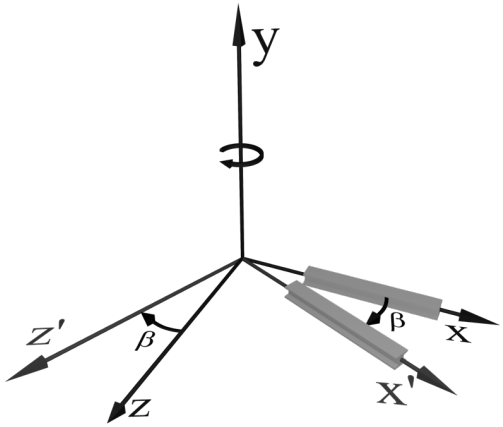


Figure 2.13: Rotation around y-axis.

In this case, we convert the original element axis x, y, z into an alternative element axis x', y, z'. Also, according to Beaufait (1971), we can write the rotation around y-axis as the matrix Eq. 2.25.

$$[R_y] = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}, \tag{2.25}$$

where β is the angle around y-axis and can be calculated using the direction cosine by the Eq. 2.26.

$$\begin{aligned} sin(\beta) &= \frac{C_x}{C_{xz}}, \\ cos(\beta) &= \frac{C_z}{C_{xz}}. \end{aligned} \tag{2.26}$$

X-Rotation

Using the principle of Spherical Coordinate System, we need only two angles(θ, ϕ) and a radius(ρ) to place an element in any position on $\mathbb{R}^3$ space. But is interesting that the rotation about the x-axis enables an additional parameter in the structural modeling. It enables the rotate the cross-section into the element axis as is illustrated in Fig. 2.14.

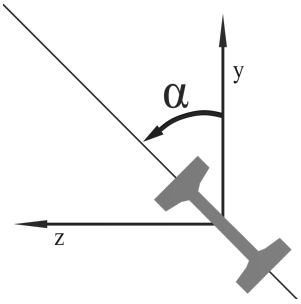


Figure 2.14: Rotation of a I-beam around x-axis.

We can write the rotation matrix about the x-axis, according to Beaufait (1971), as the matrix Eq. 2.27. The α angle is provided by the user during the structure modeling.

$$[R_x] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix}, \tag{2.27}$$

where α is the angle around x-axis.

2.3.2 3D Transformation Matrix

Given the rotation matrices of each axis in $\mathbb{R}^3$, we need to compose the transformation matrix using them.

Z-Y-X Rotation

Firstly, we need to combine the rotation matrices that were given using the rotation sequence, according to Beaufait (1971) as $[R_x][R_y][R_z]$. Performing the matrix multiplication, we can write the Z-Y-X rotation as matrix Eq. 2.28.

$$[R_1] = \begin{bmatrix} C_x & C_y & C_z \\ \frac{-C_xC_y\cos\alpha - C_z\sin\alpha}{C_{xz}} & C_{xz}\cos\alpha & \frac{-C_yC_z\cos\alpha + C_x\sin\alpha}{C_{xz}} \\ \frac{C_xC_y\sin\alpha - C_z\cos\alpha}{C_{xz}} & -C_{xz}\sin\alpha & \frac{C_yC_z\sin\alpha + C_x\cos\alpha}{C_{xz}} \end{bmatrix}. \tag{2.28}$$

Note that the rotation matrix R_1 is possible only when $C_{xz} \neq 0$, i.e. $C_x \neq 0$ or $C_z \neq 0$.

X-Z Rotation

To solve the issue when the direction cosine has only y-axis component, according to Beaufait (1971), we can use another rotation matrix for this case. To perform the rotation, note that the initial state of an element is over x-axis. To transform the element axis into direction cosine, it needs only to rotate around z-axis. Thus, we can rewrite as matrix Eq. 2.29.

$$[R_z] = \begin{bmatrix} 0 & \sin\theta & 0 \\ -\sin\theta & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.29}$$

where the $\sin\theta = C_y$.

Note that the x component is zero, thus the $\cos\theta = \cos\frac{\pi}{2} = 0$. Finally, we need to rotate the cross section around x-axis, i.e. using the matrix Eq. 2.27. According to Beaufait (1971), performing the matrix product $[R_z][R_x]$ we obtain Eq. 2.30.

$$[R_2] = \begin{bmatrix} 0 & C_y & 0 \\ -C_y\cos\alpha & 0 & \sin\alpha \\ C_y\sin\alpha & 0 & \cos\alpha \end{bmatrix}. \tag{2.30}$$

Note that if we use $\alpha = 0$, we recover the matrix Eq. 2.29.

Transformation Matrix

Using the same principle of matrix Eq. 2.16, and according to Beaufait (1971), we can write the transformation matrix Eq. 2.31.

$$[T] = \begin{bmatrix} [R] & 0 & 0 & 0 \\ 0 & [R] & 0 & 0 \\ 0 & 0 & [R] & 0 \\ 0 & 0 & 0 & [R] \end{bmatrix}, \tag{2.31}$$

where the $[R]$ can be $[R_1]$ or $[R_2]$ depending of direction cosine.

2.4 Reaction Forces

The unknowns variables presented in the global system, obtained from the stiffness method are the nodal displacements/rotations. But other properties are interesting for the analysis, as the reaction forces. Those forces can be calculated for all DOF that was set as supports in the structure modeling phase. Since we have calculated the nodal displacements/rotations a priori, the calculation of the reaction forces is a post-processing made with the global stiffness matrix using the equilibrium Eq. 2.19. We can express the reaction forces calculation using the Eq. 2.32.

$$F_i = \sum_j K_{ij} X_j, \tag{2.32}$$

where F_i is the reaction force for one DOF i, K_{ij} is a global stiffness matrix coefficient and X_j is a displacement/rotation calculated for a DOF j.

2.5 Internal Forces

The internal forces is also a post-processing using the displacements given by the stiffness method. In this case, we calculate for each node, i.e. the extremities of an element, thus, for nodes that belonging two or more elements, they can assume different values. The forces in the extremities of an element can be calculated using the Eq. 2.33.

$$f = [K^*][T][U], \tag{2.33}$$

where f is the forces on extremities of an element, K^* is the stiffness matrix of an element in local coordinates, T is the transformation matrix and $[U]$ is the displacement vector of an element.

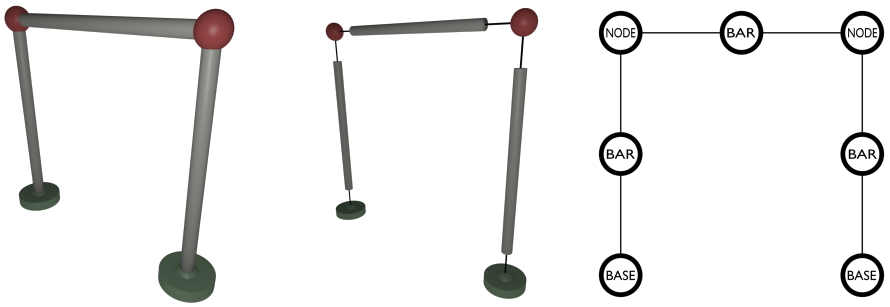
The internal forces can be calculated for an element, interpolating the forces in the extremities. Since we use only action forces exerted in the nodes of an element, we can use a linear interpolation.

3 Computational Method

In Chapter 2 we use the literature to present the frames simulation problem using the stiffness method. The resulting linear system-generated has some properties, like sparsity, positive-definite, etc. We will approach the problem using their properties to choose the best computational methods taking into optimizing the storage, performance, and precision.

3.1 Representation of the Virtual Model

The representation of the virtual model in the primary memory is made by a linked structure, but most important is that it can store the main properties of the frame structure, such as the objects that compose the virtual model and its the interconnections, and the section and material properties of the elements. However, this representation has to be capable to optimize most of the processes that need to be executed on the virtual model. Therefore, it was used as a graph structure to represent a 3D frame in the prototype. An example of a representation of the virtual model is illustrated in Fig. 3.1.



(a) 2D frame. (b) 2D frame separated into elements. (c) A graph that represents a 2D frame.

Figure 3.1: An example of 2D frame represented by a graph.

To use a graph, it is essential to define the vertex and the edge sets. For this, we will define a graph $G(V, E)$, where V is the set of all elements on the space, and E is a collection of all interconnections of these objects. The objects of V are subdivided between the objects needed to compose the structure, like nodes, bars, and bases.

A node is a mathematical entity that it is defined as a connection between two or more elements, but in the prototype a node was defined as a separated object only for modeling reasons. Another object that was added to the prototype was the base object that is a modification of a standard node, but with DOF locked, i.e. the base is the standard support.

The bar definition in the prototype is made from a combination of two elements of nodes or bases, in other words, to define a bar is necessary to create a connection between a node and a base, a base and another base, or a node and other node.

Another possibility is to use a graph with the set E as the elements and the set V as the nodes and bases, but it makes it harder to represent other components in a future project expansion, such as the slabs.

3.2 A Graph Implementation

A graph can be implemented basically by two types of structures, using an adjacency matrix or an adjacency list.

An adjacency matrix is the most efficient structure considering the access performance of write and reads operations. But if we consider the needed primary memory space, this implementation has a poor memory optimization, which in the storage of a sparse graph generates an unnecessary allocation of space using a started implementation, however, it is possible to use an optimized approach to store an adjacency matrix. Another issue of the adjacency matrix that is not flexible considering operations of adding and removing.

An adjacency list is more flexible than the adjacency matrix and is more efficient considering the storage. But this approach has problems, like information access performance.

For the implementation of the graph it was used the adjacency list approach, but with optimizations to perform the information accesses more efficient. Therefore was used index structures to assist access operations into the adjacency list. For operations that need to the search nodes of the virtual model, like the copy operation, was implemented an index based on an AVL tree.

To store the objects as a virtual model it was necessary to create an identifier that can name the objects using a serial id, thus we use this id generated on object creation

to index the node list into adjacency list.

3.3 Building the Stiffness Matrix

To the implementation of the stiffness method for the prototype, we need to define an efficient way to compute the linear system found by the combination of the transformation matrix Eq. 2.31 and the stiffness matrix Eq. 2.22 for each element. Thus, we can use some properties of the stiffness matrix to optimize the performance of the method computation. According to Bathe (1976), the stiffness matrix is strictly positive-definite, which opens up a range of the numerical method that we can use. Another property is that it is very sparse, making it possible to optimize the numerical method to not compute zero positions.

From the model created by the user, the prototype accesses the objects of the virtual model, collect the properties of the material, section, and dimensions of the elements, the state of the nodal DOF, finally, this information is used to build the linear system. It is important to consider the algorithm of construction of the global stiffness matrix and how the subsystem matrix is constructed using the boundary conditions. For

this, consider the pseudo-code presented in Alg. 1.

Algorithm 1: Build the stiffness matrix.

Data: List of model objects

Result: Stiffness matrix of the system

```
1  begin
2      create DOFCount = 0
3      for each node and base objects of the input list do
4          for each DOF of the object do
5              store the DOF as DOFCount;
6              DOFCount += 1;
7          end for
8      end for
9      for each bar object of the input list do
10         store properties;
11         create the stiffness matrix for the element with global coordinates;
12         update the global stiffness matrix using the element matrix the
           DOF of each node;
13     end for
14     for each listed DOF do
15         if displacement on DOF it was restricted then
16             remove the column of global matrix corresponding to DOF;
17             remove the row of global matrix corresponding to equilibrium
               equation of DOF;
18         endif
19     end for
20     return reduced stiffness matrix;
21 end
```

At the end of the matrix construction, taking the exerted external forces in the structure, the linear system is ready for the next phase, the solution of the linear system to obtain the displacements of the structure.

3.4 Numerical Method

Considering the strictly positive-definiteness and symmetric properties of the stiffness matrix, we can use some methods for solving the linear system. A direct method like the Cholesky factorization is used to solve systems until a determined size. After a certain system size, it becomes interesting to use iterative methods.

The Jacobi method, according to Young (1971), is an iterative method for solving the system in the form:

$$Ax = B,$$

where A is an invertible square matrix, i.e. $det[A] \neq 0$. This method can be used to compute an approximation for the solution using the Eq. 3.1.

$$(x_{k+1})_i = \frac{1}{A_{i,i}} \left(B_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{i,j}(x_k)_j \right), \forall i = 1, 2, \cdots, n; \tag{3.1}$$

where i is the ith row and j is the jth column of the matrix A. Since this is an iterative method, k represents the current iteration, and x_{k+1} is a vector of unknowns that will be used in the next iteration.

The iterative method has an advantage that we can stop the iterations following an estimated error that can be computed using the Eq. 3.2.

$$\epsilon = \frac{max|(x_{k+1})_i - (x_k)_i|}{max|(x_{k+1})_i|}, \tag{3.2}$$

where ϵ is the the required precision.

The Gauss-Seidel method is very similar to the Jacobi method, but instead of using only the unknowns variables of the previous iteration, it also uses the unknowns calculate in the same iteration. According to Young (1971), we can express this method using the Eq. 3.3.

$$(x_{k+1})_i = \frac{1}{A_{i,i}} \left(B_i - \sum_{j=1}^{i-1} A_{i,j}(x_{k+1})_j - \sum_{j=i}^{n} A_{i,j}(x_k)_j \right), \forall i = 1, 2, \cdots, n. \tag{3.3}$$

To calculate the estimated error, we can use also the Eq. 3.2.

The Successive Over-Relaxation (SOR) method is a variant of the Gauss-Seidel method, that weights the solution of the previous iteration with the corrector factor of the method. Using a parameter ω, it interleaves the solutions on each iteration. The iteration formula of the SOR method is given, according to Young (1971) by Eq. 3.4.

$$(x_{k+1})_i = (1-\omega)(x_k)_i + \frac{\omega}{A_{i,i}}\left(B_i - \sum_{j=1}^{i-1} A_{i,j}(x_{k+1})_j - \sum_{j=i}^{n} A_{i,j}(x_k)_j\right), \forall i = 1, 2, \cdots, n. \tag{3.4}$$

The method converges if the parameter is $0 < \omega < 2$. The SOR method converges faster than the Gauss-Seidel Method if an appropriate choice of the ω is made.

The Conjugate Gradient is one of the best and most used methods for solving linear systems. According to Heath (1997), the method is shown in the Alg. 2. Note that the method convergence depends if A is positive definite and if A is symmetric, which the stiffness matrix is. However, these are not the only convergence criteria.

Algorithm 2: Perform the conjugate gradient method.

Data: A, x_0, b

Result: x

begin
- $r_0 \leftarrow b - Ax_0; \quad k \leftarrow 0; \quad \beta_{-1} \leftarrow 0; \quad d_{-1} \leftarrow 0;$
- **while** $\frac{\|r_k\|}{\|b\|} \quad and \quad k < k_{max}$ **do**
  - $d_k \leftarrow r_k + \beta_k d_{k-1}$
  - $\lambda_k \leftarrow \frac{r_k^T r_k}{d_k^T A d_k}$
  - $x_{k+1} \leftarrow x_k + \lambda_k d_k$
  - $r_{k+1} \leftarrow r_k - \lambda_k A d_k$
  - $\beta_k \leftarrow \frac{\|r_k\|^2}{\|r_{k-1}\|^2}$
  - $k \leftarrow k + 1$
- **end while**
- **return** x;

end

According to Thibes (1997), the convergence rate of an iterative method depends, usually, of the spectrum of the system matrix. The spectrum of a matrix is defined as

the sets of all its eigenvalues. To improve the system matrix condition, usually, be uses a pre-conditioner method to improve the convergence rate of iterative methods, one that can be used is the incomplete Cholesky factorization.

3.4.1 Reduced Stiffness Matrix

Since the stiffness matrix is sparse, we can use the CSR (Compressed Sparse Row) method as illustrated by Fig. 3.2 to store the reduced stiffness matrix.

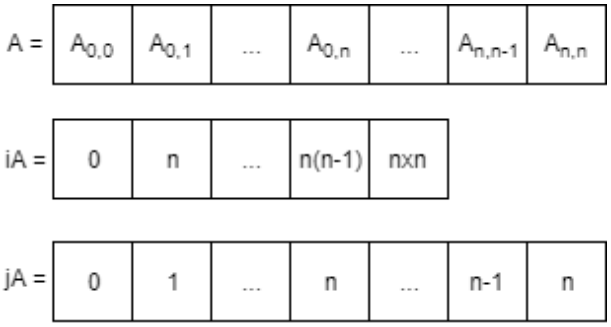


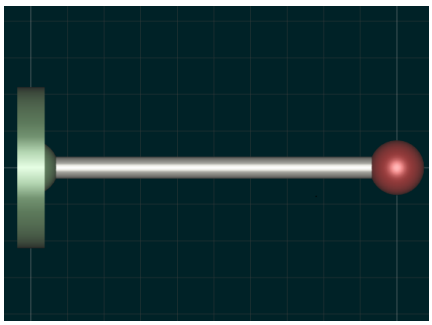
Figure 3.2: Illustration of the vectors of CSR method.

Where $A_{i,j}$ is a non-zero coefficient of the stiffness matrix, each coefficient of iA indicates the beginning of one row on A, each coefficient of jA indicates which column the coefficient of A belongs. This approach to store the stiffness matrix prevents that the solver computes zero positions. It is also possible to use the CSC (Compressed Sparse Column) approach, but it depends on the operations that will be executed into the matrix.

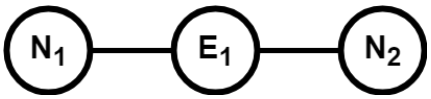
3.5 Computational Simulation Example

To demonstrate the computational simulating process of a frame structure, we will use a 2D example, composite by a beam and defined between the nodes N_1 and N_2. The node N_1 has all DOF locked. It is illustrated by Fig. 5.4.

The element namely E_1 has the property $E = 210GPa$, we consider a circular section with the moments of inertia as $I = \frac{\pi r^4}{4}$ and the section area $A = \pi r^2$. Considering the section radius $r = 0.06m$, the moments of inertia are $I = 1.01788\text{e}{-5}m^4$ and the section area $A = 0.01131m^2$. Finally, we consider all the length of the elements as $L = 10m$. All the units used was in S.I.



(a) Virtual model of a beam



(b) Representation of the virtual model into a graph

Using the element material and geometry properties defined above, we can define the stiffness matrix of the element E_1 by the Eq. 2.14. The transformation matrix is dependent on the direction vector. Therefore, the computer order of the element nodes, from N_1 to N_2 or from N_2 to N_1, it generates antiparallel vectors. Thus, if we compute the element from N_2 to N_1, the final transformation matrix is equal to the identity. However, if we invert the nodes computer order, the direction cosine is $C = (-1, 0)$, thus we obtain the transformation matrix Eq. 3.5 from the matrix Eq. 2.16.

$$T = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

Note that no matter the order of nodes computation, i.e. the transformation matrix corrects the resulting displacements independent of the direction vector sense. It is necessary to update the positions of variables IDs $(h_1, v_1, \theta_1, h_2, v_2, \theta_2)$ from matrix Eq. 2.14 to match the new order of nodes computation. For the example is defined the order of computation from N_1 to N_2, thus, the transformation matrix used is the matrix Eq. 3.5.

Using the matrix Eq. 2.14 and the properties presented above, we obtain the stiffness matrix for the element in local coordinates 3.6.

$$K^* = \begin{bmatrix} 2.37504e8 & 0 & 0 & -2.37504e8 & 0 & 0 \\ 0 & 25651 & -128252 & 0 & -25651 & -128252 \\ 0 & -128252 & 855016 & 0 & 128252 & 427508 \\ -2.37504e8 & 0 & 0 & 2.37504e8 & 0 & 0 \\ 0 & -25651 & 128252 & 0 & 25651 & 128252 \\ 0 & -128252 & 427508 & 0 & 128252 & 855016 \end{bmatrix} \tag{3.6}$$

To obtain the element stiffness matrix in global coordinates, we need to use the Eq. 2.17, where $K_G = [T^t][K^*][T]$. Thus we obtain the matrix Eq. 3.7.

$$K_G = \begin{bmatrix} 2.37504e8 & 0 & 0 & -2.37504e8 & 0 & 0 \\ 0 & 25651 & 128252 & 0 & -25651 & 128252 \\ 0 & 128252 & 855016 & 0 & -128252 & 427508 \\ -2.37504e8 & 0 & 0 & 2.37504e8 & 0 & 0 \\ 0 & -25651 & -128252 & 0 & 25651 & -128252 \\ 0 & 128252 & 427508 & 0 & -128252 & 855016 \end{bmatrix} \tag{3.7}$$

Updating the DOF IDs, we can write the system matrix Eq. 3.8.

$$\begin{bmatrix} H_2 \\ V_2 \\ M_2 \\ H_1 \\ V_1 \\ M_1 \end{bmatrix} = \begin{bmatrix} 2.37504e8 & 0 & 0 & -2.37504e8 & 0 & 0 \\ 0 & 25651 & 128252 & 0 & -25651 & 128252 \\ 0 & 128252 & 855016 & 0 & -128252 & 427508 \\ -2.37504e8 & 0 & 0 & 2.37504e8 & 0 & 0 \\ 0 & -25651 & -128252 & 0 & 25651 & -128252 \\ 0 & 128252 & 427508 & 0 & -128252 & 855016 \end{bmatrix} \begin{bmatrix} h_2 \\ v_2 \\ \theta_2 \\ h_1 \\ v_1 \\ \theta_1 \end{bmatrix} \tag{3.8}$$

Considering the node N_1 as a support with all DOF locked, exerting the boundary conditions in matrix Eq. 3.8 we obtain the matrix Eq. 3.9.

$$\begin{bmatrix} H_2 \\ V_2 \\ M_2 \end{bmatrix} = \begin{bmatrix} 2.37504e8 & 0 & 0 \\ 0 & 25651 & 128252 \\ 0 & 128252 & 855016 \end{bmatrix} \begin{bmatrix} h_2 \\ v_2 \\ \theta_2 \end{bmatrix} \tag{3.9}$$

To resolve the matrix Eq. 3.9 we define an external force as the matrix Eq. 3.10.

$$\begin{bmatrix} H_2 \\ V_2 \\ M_2 \end{bmatrix} = \begin{bmatrix} 0N \\ -1000N \\ 0N \end{bmatrix}. \tag{3.10}$$

The next step to solving the linear system is to create the auxiliary vectors described in the Section 3.4.1. The CSR representation of stiffness matrix is described by the Tab. 3.1, Tab. 3.2 and Tab. 3.3.

A	2.37504e8	25651	128252	128252	855016

Table 3.1: Non-zero coefficients of stiffness matrix.

iA	0	1	3

Table 3.2: Beginning of the rows of the stiffness matrix.

jA	0	1	2	1	2

Table 3.3: Columns of the coefficients of A.

Solving the linear system we obtain the displacements/rotations on the Eq. 3.11.

$$\begin{bmatrix} h_2 \\ v_2 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 0m \\ -0.1559m \\ 0.0233rad \end{bmatrix}. \tag{3.11}$$

The reaction forces and the internal forces of the element can be calculated using the Eq. 2.32, Eq. 2.33, and the calculated displacement.

4 Prototype Construction

The prototype is composed by five modules, *Interface*, *KLib*, *KDraw*, *KFile* and *KSim*. The modules dependencies are illustrated by the component diagram in Fig. 4.1.

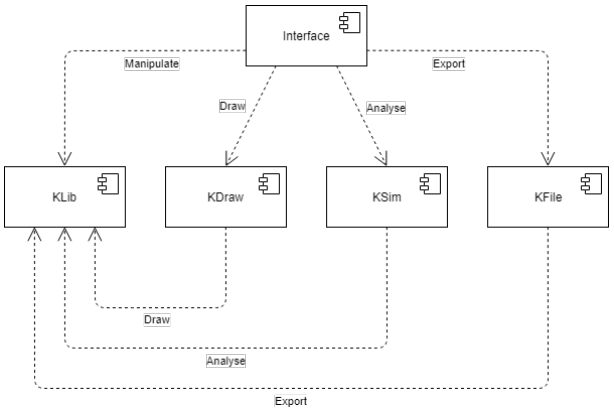


Figure 4.1: Components diagram of the prototype modules.

The interface module is written in Python language, and the other modules are written in C/C++ language.

4.1 Interface

The interface module is responsible for user experience and communication between the other modules. The user interface is inspired by commercial software of structural analysis, CAD and 3D-Modeling. The interface is illustrated in Fig. 4.2.

Python language was chosen because of the easy configuration, vast library of modules, and its portability. We use the wxPython to build the GUI, it is a version of the wxWidgets of C/C++ for Python, but easier to configure. Since we do not need hard processing of the interface, thus, an interpreted language is enough to satisfy the performance requirements.

The communication with other modules is made using Ctypes. It is a python module that enables the usage of shared objects (.so, .dll). This communication is made through of functions described in the modules header.

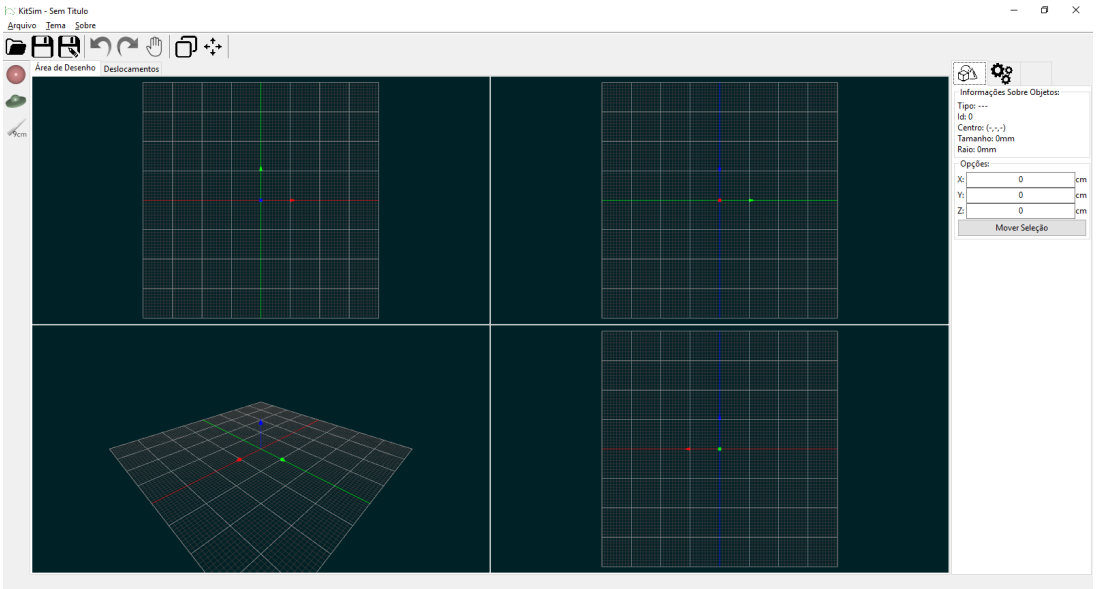


Figure 4.2: The prototype interface.

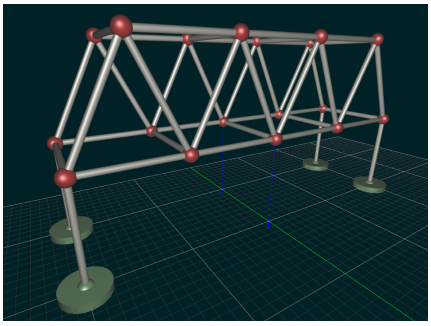
The interface enables the user to model a virtual structure using the basics transformation tools in the 3D-Modeling, translation, and rotation. Also, it was implemented the basic manipulation of forces, materials, sections, and supports of the structure. E.g., if we select a node or a base, we can change the DOF statuses, i.e., if a degree-of-freedom is locked or not.

The simulation processes return the displacements/rotations, the reaction forces, the internal forces and the displaced virtual structure to the interface. For the user can visualize the results, we implemented an option to view the displaced structure, view the displacements as color gradient, or both. The displacements can be exported for a table file (.csv). We do not have time to implement a visualization for the internal and reaction forces, however, these results are given by text mode.

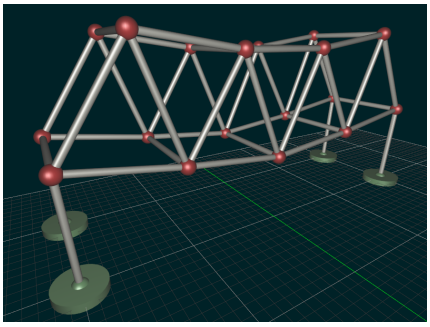
The interface enables us to save, open and create projects. It also allows exporting the project into a MATLAB file (.m), enabling the user to make an external analysis.

4.1.1 Visualization Areas

The interface has four visualization areas that the user can be alternate between the vision types, perspective, xy-plane, xz-plane and yz-plane. The visions on planes use an orthogonal camera. These visualization areas can be selected by a double-click to



(a) The original state of the 3D frame.



(b) Displaced 3D frame.

Figure 4.3: The displacement of a 3D frame.

maximize/restore one of them, thereby the user can navigate in the draw area.

The visualization areas are implemented using the canvas object of the wxPython. To we can draw in canvas object, we need to use a module called by PyOpenGL. This module is a version of the classical OpenGL implementation, but for python. In this work, the canvas object is responsible only to define the OpenGL context, camera, and light sets.

Camera

A perspective camera has the properties as the set of eye and center positions and the up vector which defines the camera rotation on space. Therefore, we use the OpenGL function *gluLookAt(eye, center, up)* to define the position, rotation, and focus of the camera. But to call the OpenGL function, we need to calculate the parameters first.

To calculate the camera eye position, we use the spherical coordinates system, where we define two angles, $0 \leq \phi \leq \pi$ and $0 \leq \theta \leq 2\pi$, and a radius ρ. θ, ϕ and ρ are illustrated in Fig. 4.4. The camera position, the eye parameter, can be calculated by Eq. 4.1.

$$\vec{E}(\theta, \phi, \rho) = (\rho(\cos\theta \sin\phi) + x_0, \rho(\sin\theta \sin\phi) + y_0, \rho(\cos\phi) + z_0), \tag{4.1}$$

where the point $F = (x_0, y_0, z_0)$ is the focus of the camera. The θ and ϕ angles are updated each time the user gives the command to rotate the camera. The user can change the zoom ratio in the visualization areas. To perform this function, the software changes the value of ρ.

The up vector indicates the up side of the camera. It can be calculated by Eq.

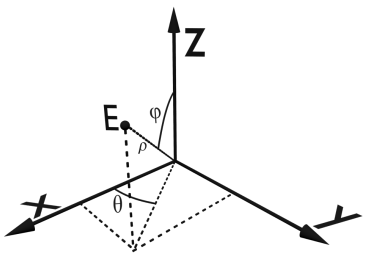


Figure 4.4: The eye position in spherical coordinates with the focus in the origin.

4.2.

$$\begin{aligned} \frac{\partial \vec{E}}{\partial \phi} &= \rho(\vec{up}) = \rho(-\cos\theta\cos\phi, -\sin\theta\cos\phi, \sin\phi) \\ \vec{up} &= (-\cos\theta\cos\phi, -\sin\theta\cos\phi, \sin\phi) \end{aligned} \tag{4.2}$$

Note that depending on the OpenGL coordinates used, the derivative can change. $\vec{up}$ is a unitary vector, thus, we need to normalize it using the length ρ.

Another function to define a perspective camera in OpenGL context is the *gluPerspective(fovy aspect, zNear, zFar)*, where *fovy* is the field of view angle, *aspect* is the aspect ratio of camera, *zNear* and *zFar* define the minimal and maximal distance that an object appear in scene.

Pan Function

One operation that can be used in the visualization areas is the pan function. This function enables the user to change the camera focus to another position. Note that exists a normal vector $\vec{N} = \vec{E} - \vec{F}$ that define a plane in $\mathbb{R}^3$, where $\vec{E}$ and $\vec{F}$ are the eye and focus position, respectively. To perform this function, the focus point is translated in this plane defined by the normal vector.

First, we create a parametric function to translate the focus point. A parametric function for a plane in $\mathbb{R}^3$ needs a point belonging to the plane, two non-collinear direction vectors and two parameters. The parameters θ and ϕ are mapped by the window coordinates x and y, using the current and last position of mouse on wxPython canvas. Using the same parameters in pan function, we can define the direction vectors $\vec{V_1}$ and $\vec{V_2}$ by Eq. 4.3.

$$\begin{aligned} \rho\vec{V_1} &= \frac{\partial\vec{E}(\theta,\phi,\rho)}{\partial\phi}, \quad \rho\vec{V_2} = \frac{\partial\vec{E}(\theta,\frac{\pi}{2},\rho)}{\partial\theta}, \\ \vec{V_1} &= \vec{up} = (-\cos\theta\cos\phi, -\sin\theta\cos\phi\sin\phi), \\ \vec{V_2} &= (-\sin\theta, \cos\theta, 0). \end{aligned} \tag{4.3}$$

In the Eq. 4.3, the direction vector $\vec{V_2}$ we use the parameter $\phi = \frac{\pi}{2}$. This is to make parallel $\vec{V_2}$ and xy-plane. The new focus position can be calculated by the Eq. 4.4.

$$\vec{F_{i+1}} = \vec{F_i} + c_1\vec{V_1}(\Delta\theta, \Delta\phi) + c_2\vec{V_2}(\Delta\theta), \tag{4.4}$$

where $\vec{F}$ is the focus position, the constants c_1 and c_2 determine the translation velocity, and i is the ith iteration. $\Delta\theta$ and $\Delta\phi$ are calculated using the current and last mouse position in window coordinates. c_1 and c_2 are two constants that determine the speed of each direction. Finally we recalculate the eye position according with the Eq. 4.1.

4.2 KLib

This module is responsible for store, in main memory, and modify the virtual model created by the user. It implements the data structures shown in Section 3.1.

The interface executes functions of this module for performing the virtual structure modifications. These modifications can be: copy of structure parts, transformation in the virtual structure or adding/removal of nodes, bases or elements.

The transformation operations that were implemented are translation and rotation. These operations are made only on the nodes and bases. Since the structural elements are defined by nodes, if we translate or rotate the nodes, the operations are made in the elements too.

To rotate a virtual structure, we can select the rotation axis, x, y or z, and the point that the axis pass through. The default point is the geometrical center of the selected objects. The rotation operation is illustrated in Fig. 4.5.

To calculate the new nodes position, given an angle θ, a node coordinate $\vec{N}$, an axis and a selected point $\vec{P}$, we start the rotation creating an auxiliary node coordinate

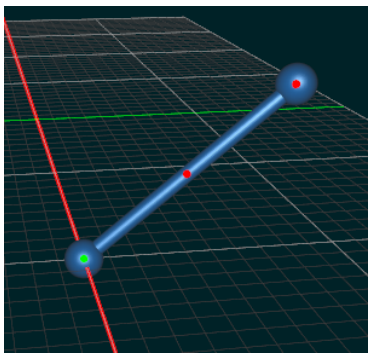


Figure 4.5: Rotation of an element in x-axis on selected point.

$\vec{N}^* = \vec{N} - \vec{P}$. We can write the rotation by matrix multiplication $N' = [N^*][R_x]$, $N' = [N^*][R_y]$ or $N' = [N^*][R_z]$, depending of the selected axis for rotation. $[R_x]$, $[R_y]$ and $[R_z]$ are the rotation matrices described in matrix Eq. 2.27, matrix Eq. 2.25, and matrix Eq. 2.23 respectively. Finally, the end coordinates of the node is given by $\vec{N} = \vec{N}' + \vec{P}$.

This module is also responsible by the undo/re-undo operations. For this, we store the the user actions into an action stack. To perform the undo/re-undo operations, we stack up and pop these actions.

4.3 KDraw

This module is responsible for drawing the virtual model that is built in *KLib*. This module also draws the visualization of displacements given by the simulation.

The interface module defines the OpenGL context, but all draw is made by this module. We use the *freeglut* library to have access to the OpenGL API. An element and a node are drawn using a cylinder and a sphere by calling the OpenGL functions *gluCylinder* and *gluSphere* respectively. The base object is also drawn using the functions described previously.

During the prototype development, we do not have time to implement an approach to reduce the draw calls. In future works, it is interesting to use the vertex array object (VAO) and the vertex buffer object (VBO) to reduce the draw calls. Note that we represent an element only using a cylinder, even having elements with different sections. In future works, it is interesting to create 3D objects with the format of the element section.

4.4 KFile

The *KFile* module is responsible for store/load and exports the virtual model into files. The main file type is *.kmp*. It is a binary file that store a header, that contains the number of objects of the virtual model, the camera position and other properties of the software state. After storing the header into *.kmp* file, we save an array that has all the objects of the virtual model.

This module provides the interface the possibility to save the displacements calculated into a *.csv* file. It also provides the possibility to export the virtual model into *.m* file, that can be used to external processing of the structure in *MATLAB*.

4.5 KSim

The last module that composes the prototype is the simulation module. This module implements the mathematical model shown in Chapter 2 and the numerical method shown in Section 3.4.

The simulation results are given to the interface using three arrays that contains the displacements, the reaction forces, and internal forces.

This module also creates a copy of the virtual model that is given to it by the interface. This copy is used to create the displaced virtual model for the user visualization.

5 Results and Discussions

To validate the mathematical and numerical models, we compare the displacements obtained from the prototype of this work with the solution obtained from SAP2000 (PORTUGAL, 2019), and from Ftool (MARTHA, 2019). We compare four methods to solve linear systems, Cholesky, Gauss-Seidel, SOR, and Conjugate Gradient. The error tolerance used is $\epsilon = 10^{-10}$. To measure the error compared with commercial software, we use the metric of the Mean Squared Error (MSE), that can be calculated as:

$$MSE = \frac{1}{N}\sum_{i=0}^{N}(X_i - \hat{X}_i)^2,$$

where X is the exact solution, and $\hat{X}$ is the approximated solution. There is a way to measure the difficulty to solve a linear system equations $Ax = b$, the condition number. The condition number can be calculated as

$$cond[A] = \|A\|_\infty \left\|A^{-1}\right\|_\infty,$$

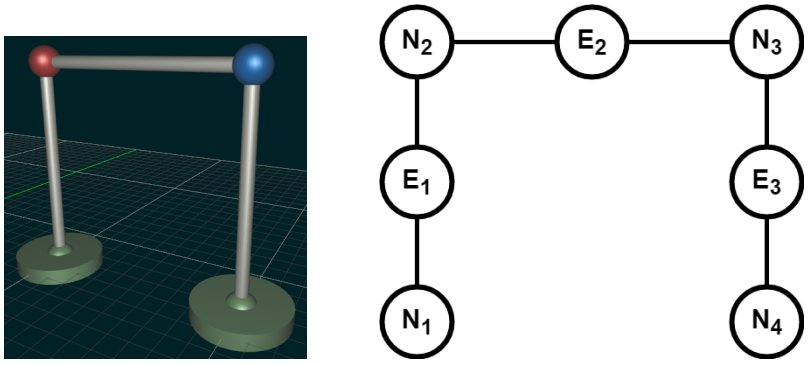
where the matrix norm $\|A\|_\infty = max\{\sum_j |A_{ij}|, \forall i\}$, being i the ith row and j the jth column of A.

According to Heath (1997), a problem is said to be insensitive, or well-conditioned, if a given relative change in the input data causes a reasonably commensurate relative change in the solution. A problem is said to be sensitive, or ill-conditioned, if the relative change in the solution can be much larger than that in the input data. The smaller $cond[A]$ is, most well-conditioned A is.

For all validation tests, we use $E = 210Gpa, \nu = 0.38522$, and the circular cross section with radius $r = 0.06m$.

5.1 Model 1

The first test we compare the solutions of the model be illustrated in Fig. 5.1.



(a) Virtual model of a 2D frame.

(b) Representation of the virtual model into a graph.

Figure 5.1: Model 1: 2D frame validation.

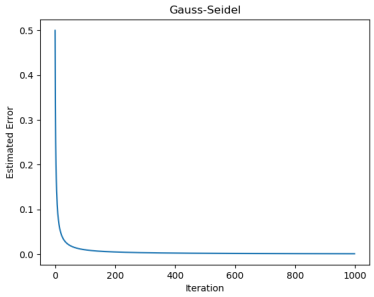
The nodes N_1 and N_4 are the supports of the structure, i.e. we assume that the displacements/rotations are null. For all elements, the length $L = 10m$. We exerted an external force $F = (1KN, 0, 0)$ in the node N_3 that is highlighted as blue in Fig. 5.1. To process a 2D structure into a 3D analysis, we need to set additional boundary conditions on the model, thus, we set the nodes unknowns $T_y = R_x = R_z = 0$, i.e. we return to the analysis presented in Section 2.2. The progression of error through the iterations of the methods are illustrated by Fig. 5.2.

Note that the number of iterations of Conjugate Gradient method is less than other approaches. That is can be observed by Tab. 5.1.

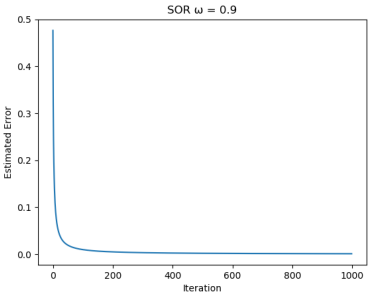
Note that in Fig. 5.3b and in Fig. 5.3c, the commercial software extrapolated the displacements, they made it only for visual representation. In Fig. 5.3a the prototype plots the real displacements, but to assist the visualization of the model we implement a color gradient. The red nodes are the ones that have the largest displacements and the blue nodes are ones that have the null displacements.

We can observe that the error behavior in the first 1000 iterations of the Gauss-Seidel and SOR methods are very close. The difficulty of both methods is the solution fine adjustment to achieve the specified error.

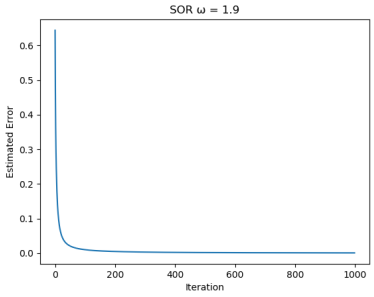
Assuming the solution obtained from SAP2000 and Ftool as exact, we solve the linear equations we use a direct method, the Cholesky Factorization, to compare



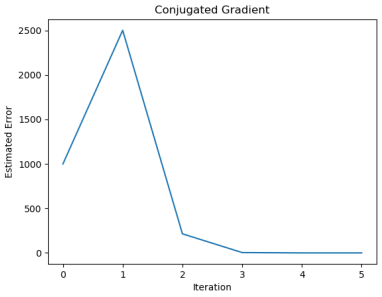
(a) First 1000 iterations of Gauss-Seidel method.



(b) First 1000 iterations of SOR method using $\omega = 0.9$.



(c) First 1000 iterations of SOR method using $\omega = 1.9$.



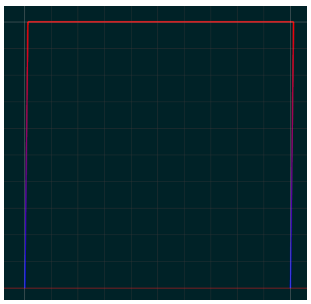
(d) The behavior of all iterations of Conjugate Gradient method.

Figure 5.2: The behavior of the error by the interactions of the methods.

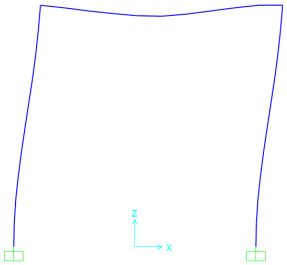
	Iterations
Gauss-Seidel	355466
SOR ($\omega = 0.9$)	427644
SOR ($\omega = 1.9$)	22952
Conjugate Gradient	5

Table 5.1: Number of iterations of the methods for Model 1.

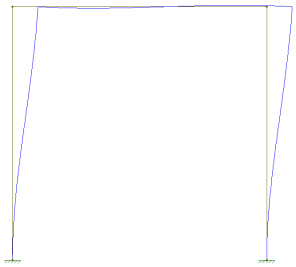
the results. Finally, we find the MSE $1.66e-10$ comparing the solution obtained from Cholesky method and solution given by SAP2000. The MSE obtained from solution from Cholesky method and solution given by Ftool is $1.66e-11$.



(a) Model 1 displaced by prototype and using color interpolation.



(b) Model 1 displaced by SAP2000.

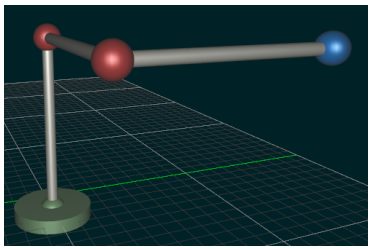


(c) Model 1 displaced by Ftool.

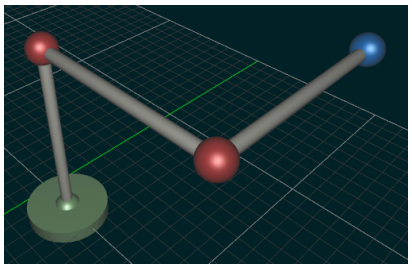
Figure 5.3: Model 1 displacements given by software.

5.2 Model 2

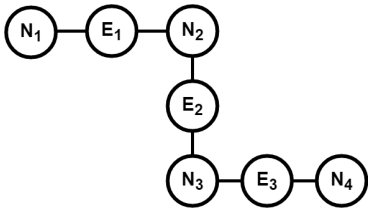
Another model that we compare is the model illustrated in Fig. 5.4. To this model can not be possible to use the Ftool to compare the results, since it does not perform 3D analysis, thus, we use only the SAP2000. For all elements, the length $L = 10m$. We exerted an external force $F = (0, 0, -1KN)$ in the node N_4 that is highlighted as blue in Fig. 5.4. The node N_1 is the base represented in Fig. 5.4a. We use it as support, i.e. we assume that all its DOF are null.



(a) Virtual model of a column with two beams.



(b) Virtual model of a column with two beams.



(c) Virtual model represented by a 2D graph.

Figure 5.4: Virtual model of a column with two beams.

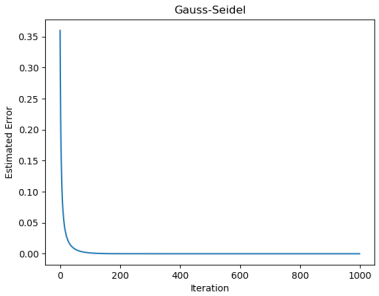
If we compute the condition number of the stiffness matrix of this model, we find $cond[S] = 1121633$. Thus, we can note that the global stiffness matrix S for this model

is ill-conditioned. This affects the convergence rate of iterative methods, such as we can observe in Tab. 5.2.

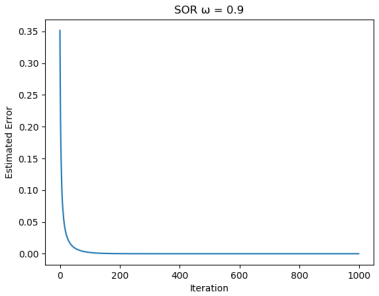
	Iterations
Gauss-Seidel	1960182
SOR ($\omega = 0.9$)	2339848
SOR ($\omega = 1.9$)	138491
Conjugate Gradient	48

Table 5.2: Number of iterations of the methods for the Model 2.

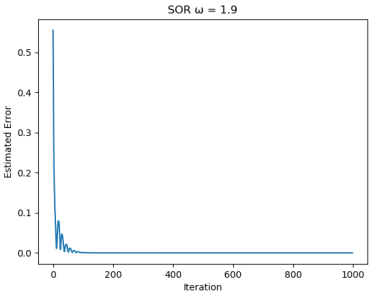
Note that in Fig. 5.5c and in Fig. 5.5d the estimated error presents instability, but it does not affect the execution of the method. Once again, the Conjugate Gradient converged faster than other iterative methods as we expected.



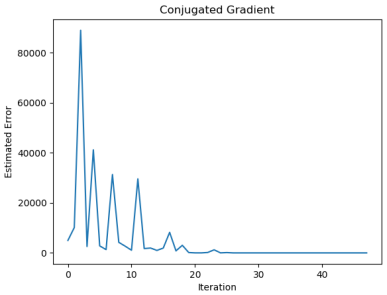
(a) First 1000 iterations of Gauss-Seidel method.



(b) First 1000 iterations of SOR method using $\omega = 0.9$.



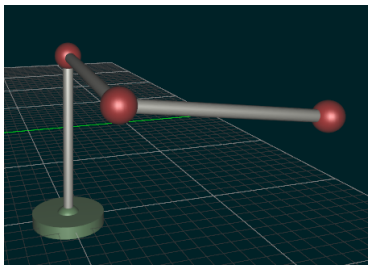
(c) First 1000 iterations of SOR method using $\omega = 1.9$.



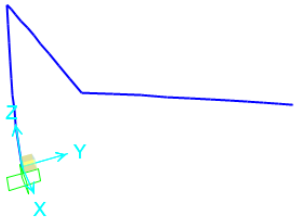
(d) The behavior of all iterations of Conjugate Gradient method.

Figure 5.5: The behavior of the error through the methods interactions.

In fact, the SOR method converges faster than Gauss-Seidel if we use the $\omega = 1.9$. The both methods still to present difficulty to do the fine adjustment to the stipulated error. For all tests we use the $\Delta\omega = 0.1$ starting with $\omega = 0.1$ until $\omega = 1.9$. We observe that the convergence rate increases when ω increases. This is a feature that we note in all tests, but it is not applicable for linear systems in general.



(a) Model 2 displaced by prototype.



(b) Model 2 displaced by SAP2000.

Figure 5.6: Model 2 displacements given by software.

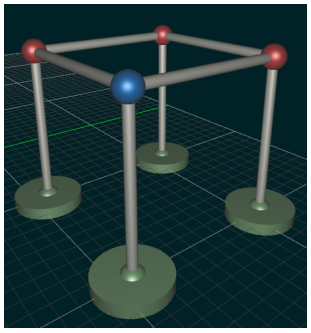
	SAP2000
Cholesky	1.52e-5

Table 5.3: MSE between the solution from Cholesky method and that obtained from SAP2000.

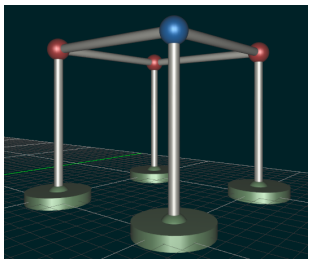
Using the displacements obtained from Cholesky Factorization, we calculate the MSE described in Tab. 5.3.

5.3 Model 3

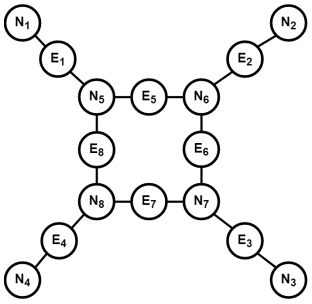
The next model that we validate is illustrated in Fig. 5.7, we use the nodes N_1, N_2, N_3 and N_4 as support. Note that N_1, N_2, N_3 and N_4 are the illustrated as bases. For all elements, the length $L = 10m$. We exerted an external force $F = (40KN, 0, 0)$ in node N_8 that is highlighted in Fig. 5.7. The condition number of the stiffness matrix is $cond[S] = 25561$.



(a) A vision of the 3D frame.



(b) A vision of the 3D frame.



(c) A graph that represent the 3D frame.

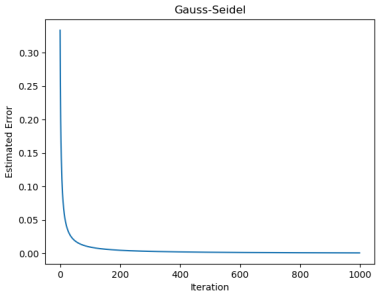
Figure 5.7: Virtual Model for a 3D frame.

Even though the linear system of the Model 3 is larger than the Model 2, the iterations number shown in Tab. 5.4 is less than shown in Tab. 5.2. This happens because

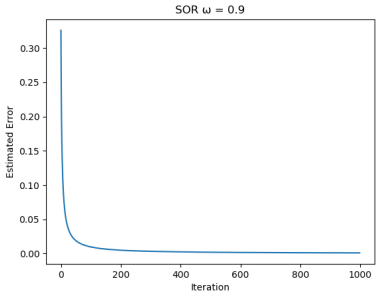
	Iterations
Gauss-Seidel	349419
SOR ($\omega = 0.9$)	420334
SOR ($\omega = 1.9$)	138491
Conjugate Gradient	40

Table 5.4: Number of iterations of the methods for the Model 2.

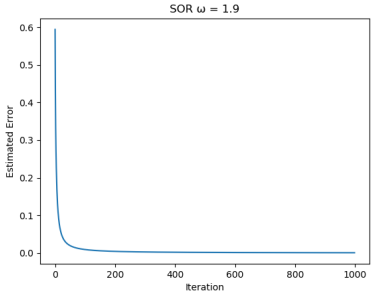
of the condition number of the Model 3 linear system is smaller than the Model 2.



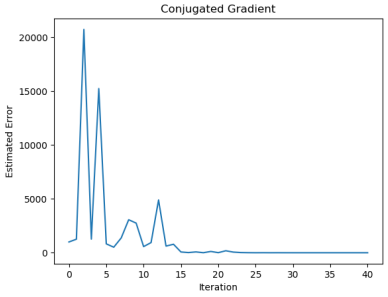
(a) First 1000 iterations of Gauss-Seidel method.



(b) First 1000 iterations of SOR method using $\omega = 0.9$.



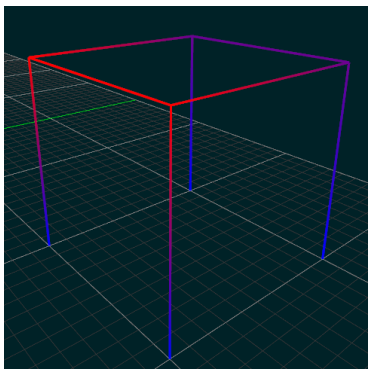
(c) First 1000 iterations of SOR method using $\omega = 1.9$.



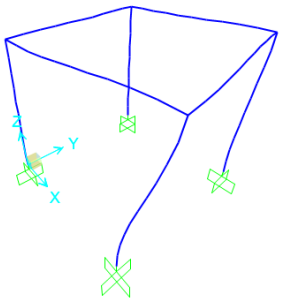
(d) The behavior of all iterations of Conjugate Gradient method.

Figure 5.8: The behavior of the error through the methods interactions.

Using the displacements obtained from Cholesky Factorization, we calculate the MSE $= 1.054e - 3$ comparing with the solution given by SAP2000.



(a) Model 3 displaced by prototype.



(b) Model 3 displaced by SAP2000.

Figure 5.9: Model 3 displacements given by software.

5.4 Model 4

The last model that we use to validate the methods, its have 80 nodes and 160 elements. The horizontal elements have length $L = 20m$ and the vertical elements have length $L = 10m$. All the bases illustrated in Fig. 5.10 are considered as supports with their DOF as null. We exerted an external force into node $N_1, \vec{F} = (40KN, 0, 0)$ that is the highlighted node as blue in Fig. 5.10. The condition number of the stiffness matrix is $cond[S] = 649267$.

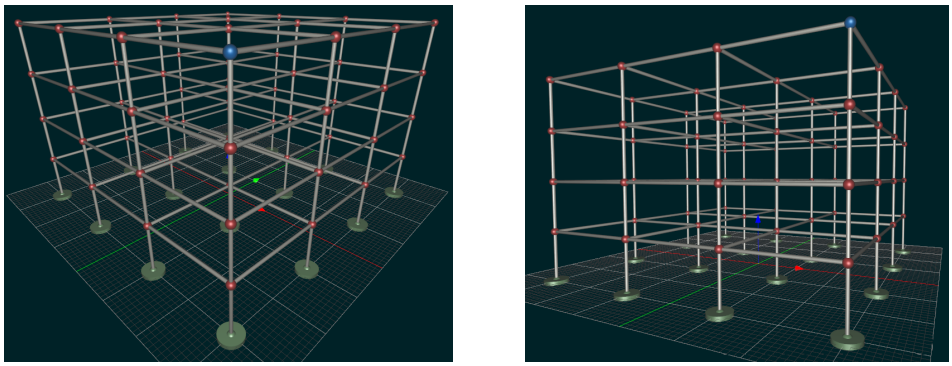


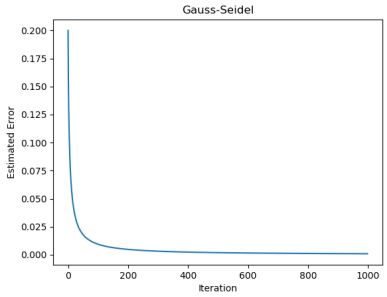
Figure 5.10: Virtual model of a complex structure composed by 3D frames.

	Iterations
Gauss-Seidel	921846
SOR ($\omega = 0.9$)	1106738
SOR ($\omega = 1.9$)	61084
Conjugate Gradient	594

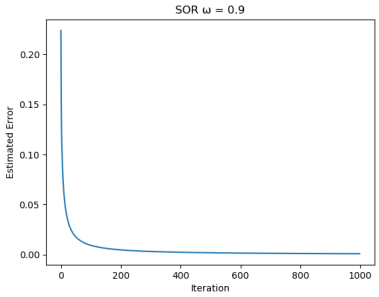
Table 5.5: Number of iterations of the methods for the Model 4.

The iterations cost of Gauss-Seidel/SOR is less than the Conjugate Gradient

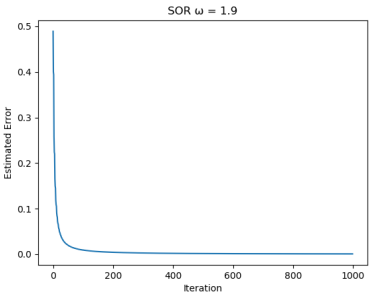
method, but the iterations differences on all test made the Conjugate Gradient method more efficient of the ones that we use.



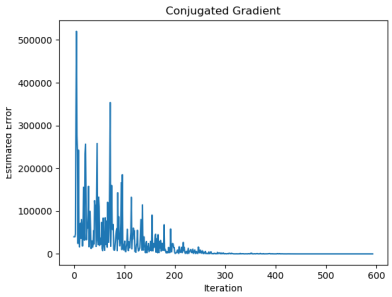
(a) First 1000 iterations of Gauss-Seidel method.



(b) First 1000 iterations of SOR method using $\omega = 0.9$.

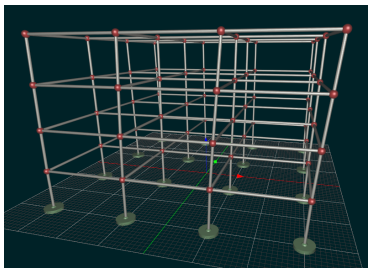


(c) First 1000 iterations of SOR method using $\omega = 1.9$.

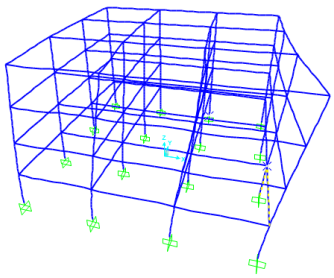


(d) The behavior of all iterations of Conjugate Gradient method.

Figure 5.11: The behavior of the error through the methods interactions.



(a) Model 4 displaced by prototype.



(b) Model 4 displaced by SAP2000.

Figure 5.12: Model 4 displacements given by software.

The MSE calculated for the result obtained from Cholesky Factorization and solution from SAP2000 is 0.1306. Note that the MSE calculated for this model is significantly bigger than other models. If we consider that largest displacement of this model occurs in N_1 with displacement calculated by prototype:
$\Delta N_1 = (2.99269, 0.6047, -1.238e-4)$. The node N_1 is also one with biggest error if we

compare with SAP2000 solution: $\Delta N_1 = (2.99315, 0.60489, -1.248e-4)$. Since we use a linear approach to do the 3D analysis, the mathematical model is being subjected to the Small Displacements Hypothesis that we assume in the introduction.

We can observe that the implemented Gauss-Seidel and SOR methods are sensitivity to the matrix condition number, which can need an exaggerated number of steps to converges to the solution. The Conjugate Gradient have less sensitivity to the condition number.

6 Conclusion

In this work, we discussed the linear approach to analyze 3D frames based at Stiffness Method. We presented an introductory chapter about structure analysis, that contains the theory behind the linear analysis of structures, their properties and most important objects of study. We also discussed the representation in main memory and storage of the virtual structure, numerical methods to solve linear systems and how to build the problem in software. The main aim of this work was to create a prototype that would enable the user to build a virtual model of a structure and analyze it, and we do this.

The developed prototype enables the user to create a virtual model that represents a real structure. It also enables the user to determine many support types, material properties, and section geometry. We also can exert an external force in any node of the structure.

We validate the mathematical method in Chapter 5, the results obtained using the Cholesky Factorization from the linear system are satisfactory. The largest calculated error is in Model 4, where we calculate the $MSE = 0.1306$. But if we analyze the biggest relative error between our solution and the SAP2000 solution, we obtained a precision in the order of $1e-3m$, i.e. a millimetric precision. Note that the mathematical method is subject to the small displacement hypothesis, which we assume in Chapter 1. The largest displacement in Model 4 is in order of $2.99m$, that if we compare to the structure size, is a big displacement, thus, it is generated an imprecision in the result.

In fact, the Conjugate Gradient has a nice performance in this work if we compared with other methods, which was expected. But we can improve the convergence rate if we implement a preconditioner.

As future works, we intend to include another type of analysis into the prototype, like non-linear analysis, the structural dynamics, and the possibility of element refinement using the finite-elements principle.

Bibliography

BATHE, E. L. W. K. J. *Numerical Methods in Finite Element Analysis*. [S.l.]: Prentice Hall, 1976.

BEAUFAIT, F. W. *Computer Methods of Structural Analysis*. [S.l.]: Prentice Hall, 1971.

HEATH, M. T. *Scientific Computing: An Introductory Survey*. [S.l.]: The McGraw-Hill Companies, 1997.

KATTAN, P. I. *MATLAB Guide to Finite Elements: An Interactive Approach*. [S.l.]: Springer, 2007.

MARTHA, L. F. *Two-dimensional Frame Analysis Tool*. 2019. Disponível em: ⟨https://www.ftool.com.br⟩.

MCCORMAC, J. K. N. J. C. *Structural Analysis Using Classical and Matrix Methods*. [S.l.]: LTC, 2009.

PORTUGAL, C. . S. I. *SAP2000: Programa de elementos finitos para modelação, análise e dimensionamento de qualquer tipo de estrutura*. 2019. Disponível em: ⟨https://www.csiportugal.com/software/2/sap2000⟩.

RETHWISCH, W. D. C. J. D. G. *Materials Science and Engineering*. [S.l.]: John Wiley and Sons, 2007.

THIBES, H. V. *Um Estudo da Fatoração Incompleta LU e Cholesky como Pré-Condicionadores nos Métodos Iterativos*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 1997.

WEAVER, J. W. *Matrix Analysis of Framed Structures*. [S.l.]: Van Nostrand Reinhold Company, 1980.

YOUNG, D. M. *Iterative Solution of Large Linear Systems*. [S.l.]: Academic Press, 1971.