UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
DEPARTAMENTO DE ENGENHARIA DAS CONSTRUÇÕES E ESTRUTURAS
CURSO DE ENGENHARIA CIVIL

D'JEFFERSON SMITH SANTOS MARANHÃO

DESENVOLVIMENTO DE FERRAMENTA COMPUTACIONAL PARA ANÁLISE DE ESTRUTURAS RETICULADAS PLANAS

São Luís
2017

D'JEFFERSON SMITH SANTOS MARANHÃO

DESENVOLVIMENTO DE FERRAMENTA COMPUTACIONAL PARA ANÁLISE DE ESTRUTURAS RETICULADAS PLANAS

Monografia apresentada ao Curso de Engenharia Civil da Universidade Estadual do Maranhão – UEMA como requisito para a obtenção do grau de bacharel em Engenharia Civil.

Orientador: Prof. Me. Carlos César Pereira de Almeida

São Luís

2017

Maranhão, D'jefferson Smith Santos.
Desenvolvimento de ferramenta computacional para análise de estruturas reticuladas planas / D'jefferson Smith Santos Maranhão. – São Luís, 2017.
119f.

Monografia (Graduação) – Curso de Engenharia Civil, Universidade Estadual do Maranhão, 2017.

Orientador: Prof. Carlos César Pereira Almeida.

1. Análise de estruturas. 2. Métodos de elementos finitos. 3. Métodos numéricos. 4. Python. I. Título.

CDU 624.04:004.4

A meus pais, João Maranhão e Goretti Maranhão, e a minha família, pelo incentivo, amor e compreensão.

AGRADECIMENTOS

A Deus, por ter me permitido alcançar mais esta vitória.

Ao Prof. Carlos César, pela confiança e dedicação na orientação deste trabalho.

Aos meus pais, João e Goretti, pelo apoio incondicional ao longo desta jornada.

Aos meus irmãos, pelo amparo nos momentos de angústia e aflição.

A minha namorada, pelo carinho e pela compreensão, mesmo durante minhas ausências.

Aos meus colegas de curso, pelo companheirismo e pela troca de conhecimentos.

A todos os professores e funcionários do Curso de Engenharia, pela paciência e apoio ao longo dos últimos anos.

A todos que, de alguma forma, colaboraram, direta ou indiretamente, para a realização deste trabalho, registro o meu muito obrigado.

RESUMO

Trabalho acerca do desenvolvimento de ferramenta computacional para análise de estruturas reticuladas planas. A análise de estruturas é a etapa responsável por idealizar o comportamento da estrutura, sendo de enorme relevância para a concepção de um projeto de engenharia. De modo geral, a análise de estruturas tem a finalidade precípua de determinar forças internas e externas, deslocamentos, tensões e deformações, levando em consideração os possíveis estágios de carregamentos e solicitações a que a estrutura está submetida. Se realizados manualmente, os cálculos envolvidos nesta etapa podem se tornar extremamente complexos e dispendiosos, podendo ser necessária a utilização de ferramentas computacionais para a obtenção desses resultados. Contudo, tais ferramentas nem sempre estão disponíveis a custos acessíveis ou possuem a rotina de cálculo desejada. Estes fatores são determinantes para que o engenheiro se motive a desenvolver as próprias ferramentas, que, embora mais simples, contemplem as particularidades almejadas. Nesse sentido, este trabalho apresenta os conceitos do método dos elementos finitos voltados à análise de estruturas reticuladas planas (treliças, vigas e pórticos), por meio do desenvolvimento de uma ferramenta computacional, codificada na linguagem de programação Python, com o auxílio de bibliotecas de códigos para aplicações científicas (NumPy, ScyPi e MatPlotLib). Em sua primeira versão, o aplicativo desenvolvido é capaz de processar os elementos individuais de uma estrutura, realizar a análise estrutural, exibir os diagramas de esforços internos e apresentar as reações dos apoios.

ABSTRACT

Work on the development of a computational tool for the analysis of plane reticulated structures. The structural analysis is the stage responsible for idealizing the behavior of a structure, having a great importance for the design of an engineering project. In general, the structural analysis has the primary purpose of determining internal and external forces, displacements, tensions and deformations, taking into account the possible stages of loads and requests to which the structure is submitted. If performed manually, the calculations involved in this step can become extremely complex and slow, and it may be necessary to use computational tools to obtain these results. However, such tools are not always available at affordable costs or have the desired calculation routine. These factors are crucial for the engineer to motivate himself to develop his own tools, which, although simpler, contemplate the particularities desired. In this sense, this work presents the concepts of the finite element method for the analysis of plane reticulated structures (trusses, beams and frames), through the development of a computational tool, coded in the Python programming language, with the aid of code libraries for scientific applications (NumPy, ScyPi and MatPlotLib). In its first version, the developed application is able to process the individual elements of a structure, perform the structural analysis, display the internal stress diagrams and show the support reactions.

SUMÁRIO

1. INTRODUÇÃO

A análise estrutural possui enorme relevância na concepção de um projeto, pois é nesta etapa que se idealiza o comportamento da estrutura. De modo geral, a análise de estruturas tem como finalidade precípua a determinação de forças internas e externas, deslocamentos, tensões e deformações, levando-se em consideração os possíveis estágios de carregamentos e solicitações a que está submetida a estrutura.

Segundo Pereira (2011), os métodos já consagrados, tais como o método das forças e o método dos deslocamentos, constituem a base da teoria de análise de estruturas. Estes métodos, combinados com os desenvolvimentos da análise matricial de estruturas, serviram de substrato para a criação do método dos elementos finitos, que tem sido amplamente utilizado para o estudo de estruturas reticuladas planas.

Conforme explica Zienkiewicz et al. (2000), a mente humana é incapaz de compreender a complexidade de um ambiente e recriá-la em uma única operação. Por isso, o processo de subdividir o sistema em componentes foi a maneira que engenheiros, cientistas e economistas encontraram para entender o comportamento de todo um sistema a partir de elementos de compreensão mais fácil.

O método dos elementos finitos baseia-se exatamente nesta conclusão, que faz uso da estratégia de dividir e conquistar. Segundo Bathe (2014), este método tem sido amplamente utilizado no campo das análises de engenharia, onde pode ser empregado na análise de sólidos, estruturas, transferência de calor e fluidos.

Neste trabalho, pretende-se descrever uma abordagem do método dos elementos finitos voltada à resolução de estruturas reticuladas planas, tais como treliças, vigas e pórticos, bem como apresentar uma implementação computacional deste método por meio da linguagem de programação Python, com o apoio das bibliotecas científicas NumPy, ScyPy e MatPlotLib.

1.1 Delimitação do tema e formulação do problema de pesquisa

A análise estrutural é a etapa responsável pela concepção do cálculo de estruturas complexas, tais como edifícios, pontes, elevados e barragens. Os cálculos envolvidos nessa etapa, quando realizados de forma manual, podem se tornar extremamente complexos e dispendiosos. Daí, surge a necessidade de se utilizar ferramentas computacionais para a obtenção desses resultados.

A maioria das ferramentas computacionais existentes no mercado baseiam-se no método de elementos finitos. Tais sistemas possibilitam a análise dos mais variados tipos de estruturas, bem como a obtenção de informações acerca de deformações, esforços, deslocamentos e tensões.

Todavia, o fator custo tem se apresentado como um grande impeditivo para diversos profissionais da área, pois nem sempre a aquisição deste tipo de ferramenta se mostra viável. Outra dificuldade reside na elaboração de cálculos muito específicos, que nem sempre podem ser realizados por meio destas ferramentas, vez que estas apresentam rotinas padronizadas.

Neste cenário, surgiram os seguintes questionamentos: admitindo-se a premissa de que a ferramenta computacional deve apresentar baixo custo, é possível criar um sistema informatizado, baseado no método dos elementos finitos, para a análise de estruturas reticuladas planas? Caso seja possível, quais as dificuldades envolvidas no processo de desenvolvimento deste tipo de aplicativo?

1.2 Objetivos

Com base nas informações obtidas em pesquisa bibliográfica, pretende-se aplicar os conceitos do método dos elementos finitos à análise de estruturas reticuladas planas (treliças, vigas e pórticos), por meio do desenvolvimento de uma ferramenta computacional, codificada na linguagem de programação Python, que fará uso de bibliotecas de códigos para aplicações científicas (NumPy, ScyPi, MatPlotLib e ReportLab).

Como uma forma de validar os resultados gerados pela ferramenta, far-se-á a comparação de seus relatórios analíticos com os dados numéricos exportados pelo sistema educacional denominado de Ftool 3.01 (um programa gráfico-interativo para o ensino de comportamento de estruturas).

1.2.1 Objetivo geral

Desenvolver uma ferramenta computacional, com fundamento no método dos elementos finitos, para análise de estruturas reticuladas planas, principalmente, treliças, vigas e pórticos.

1.2.2 Objetivos específicos

a) Aplicar os conhecimentos adquiridos acerca do método dos elementos finitos à análise de estruturas reticuladas planas, para a criação de uma ferramenta computacional, escrita na linguagem Python, que possibilite a obtenção de deslocamentos, esforços e deformações de modo simplificado;

b) Facilitar a entrada de dados referentes aos elementos da estrutura, isto é, as coordenadas de nós; as características de cada barra; os carregamentos a que se submete a estrutura; as configurações dos apoios e articulações; e as propriedades dos materiais (módulo de elasticidade, área da seção transversal da barra e momento de inércia);

c) Determinar os deslocamentos nodais; as reações nodais; os esforços e as deformações em cada barra; e

d) Comparar os resultados analíticos gerados pela ferramenta com os dados numéricos exportados pelo sistema educacional denominado de Ftool 3.01.

1.3 Justificativa

A etapa de cálculo dos esforços e deslocamentos de uma estrutura, se realizada manualmente, pode se tornar demasiadamente complexa e cansativa. Por isso, geralmente quando o cálculo analítico é elaborado de forma manual, acaba ficando restrito a estruturas planas mais simples.

Para análises mais complexas, normalmente se utiliza uma ferramenta informatizada baseada no método dos elementos finitos. O poder computacional necessário para a análise de estruturas planas varia de acordo com a complexidade de suas geometrias e, portanto, com o tamanho das matrizes envolvidas neste procedimento.

Neste contexto, surge a necessidade de o profissional de engenharia procurar por sistemas disponíveis no mercado, que nem sempre se mostram viáveis por conta dos altos custos envolvidos na aquisição de ferramentas deste tipo. Fora isso, os programas disponíveis nem sempre contemplam todas as funções específicas de cálculo estrutural que o profissional necessita.

Estes fatores são determinantes para que o profissional da área se motive a desenvolver suas próprias ferramentas, que, apesar de mais simples, atendem às particularidades desejadas.

Por conta disso, pretende-se desenvolver uma ferramenta computacional, baseada no método dos elementos finitos, que possibilite a análise de estruturas reticuladas planas, em regime estático, e, ao mesmo tempo, consiga ser simples e atender às restrições de qualidade e custos impostas.

2. REFERENCIAL TEÓRICO

2.1 Estruturas reticuladas planas

Segundo Süssekind (1981), as estruturas compõem-se de uma ou mais peças, ligadas entre si e ao meio exterior de modo a formar um conjunto estável, ou seja, um conjunto capaz de receber solicitações externas, absorvê-las internamente e transmiti-las até seus apoios, onde estas solicitações encontrarão seu sistema estático equilibrante.

Denomina-se estrutura reticulada àquela que pode ser representada por quatro tipos de elementos: nós, barras, rótulas (ou articulações) e apoios.

Os nós representam os pontos de interseção dos eixos de barras adjacentes, podendo ou não ter uma representação física na estrutura real.

As barras são elementos estruturais prismáticos em que a dimensão longitudinal é muito superior às suas dimensões transversais, como as vigas e os pilares. Neste trabalho, admite-se a hipótese simplificadora de que as barras têm eixo reto e seção constante. Entretanto, destaca-se que as barras curvas podem ser aproximadas por um conjunto de segmentos retos e que as barras de seção variável podem ser representadas por um conjunto de segmentos de seção constante.

As rótulas, ou articulações, são sistemas que possibilitam o surgimento de rotações relativas entre as seções transversais das barras, podendo representar uma idealização de cálculo ou elementos construtivos concebidos para este efeito.

Por fim, os apoios são sistemas que impedem, total ou parcialmente, os deslocamentos dos nós de extremidade de uma barra ligada à fundação, podendo representar uma idealização de cálculo ou um elemento construtivo específico. O movimento que está sendo impedido provoca o surgimento de um esforço que determinará a reação transmitida à fundação.

2.2 Modelos estruturais básicos

Segundo Pereira (2011), a análise de estruturas compreende o estudo de cinco modelos estruturais básicos: treliça plana; treliça espacial; pórtico plano, em que se inclui o caso particular das vigas; pórtico espacial; e grelha plana.

O presente trabalho tem como campo de estudo as estruturas reticuladas planas e versará exclusivamente sobre treliças, vigas e pórticos planos.

A treliça plana, de acordo com Martha (2010), é uma estrutura reticulada que tem todas as ligações entre barras articuladas, isto é, as barras podem girar independentemente nas ligações. O comportamento da estrutura se dá fundamentalmente a esforços internos axiais, visto que os esforços cortantes e momentos fletores são pequenos se comparados com os esforços normais.

O pórtico plano, conforme o supracitado autor, é um modelo plano de uma estrutura tridimensional, podendo corresponder a uma fatia da estrutura, ou representar uma simplificação do comportamento tridimensional. Estruturas deste tipo estão contidas em um plano e as cargas também estão contidas neste mesmo plano. Isso inclui forças com componentes nos eixos x e y e momentos em torno do eixo z (normal ao plano).

As vigas planas, por sua vez, são elementos lineares em que a flexão é preponderante. Portanto, os esforços predominantes nestas estruturas são o momento fletor e a força cortante. As vigas têm a função estrutural de transmitir as cargas de paredes e lajes para os pilares, que trabalham principalmente à compressão.

2.3 Sistemas de coordenadas

Neste trabalho foram utilizados dois sistemas de coordenadas diferentes para o cálculo dos esforços e deslocamentos. O primeiro, que está relacionado com a estrutura propriamente dita, é denominado de sistema de coordenadas global. O outro, que está vinculado aos elementos estruturais (p.ex. uma barra de um pórtico), é denominado sistema de coordenadas local.

Segundo Martha (1993), os sistemas de coordenadas não devem ser confundidos com o sistema de eixos. As coordenadas globais são geralmente definidas segundo um sistema de eixos global da estrutura, já as coordenadas locais podem ser definidas tanto no sistema de eixos global quanto em um sistema de eixos local particular de um elemento estrutural.

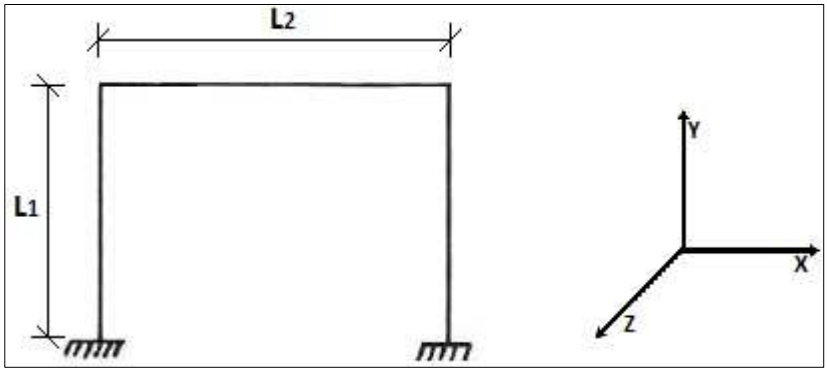
2.3.1 Coordenadas globais

De acordo com Martha (1993), as coordenadas globais são usadas para descrever uma dada configuração deformada da estrutura, ou para descrever um grupo de forças aplicadas à estrutura.

A descrição das forças aplicadas à estrutura e de seus deslocamentos fica limitada aos pontos em que são definidas as coordenadas. As informações quanto aos deslocamentos e esforços em outros pontos são obtidas em função das informações nas coordenadas globais.

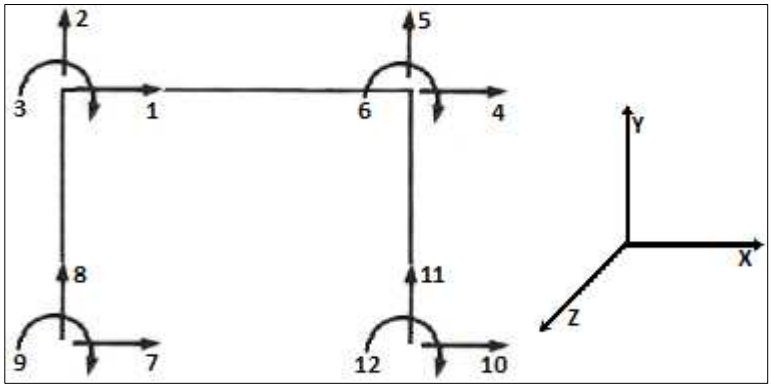
Levando-se em consideração o pórtico plano mostrado na Figura 01, suas coordenadas globais podem ser definidas conforme apresentado na Figura 02, em que foram desprezadas as condições de apoio da estrutura.

Figura 01 – Pórtico plano



Fonte: Martha (1993)

Figura 02 – Coordenadas globais



Fonte: Martha (1993)

A configuração deformada da estrutura da Figura 2 é definida pelos deslocamentos nas coordenadas globais descritos pelo vetor D formado por 12 componentes, sendo cada componente D_i referente ao deslocamento da coordenada i. De forma análoga, o grupo de forças

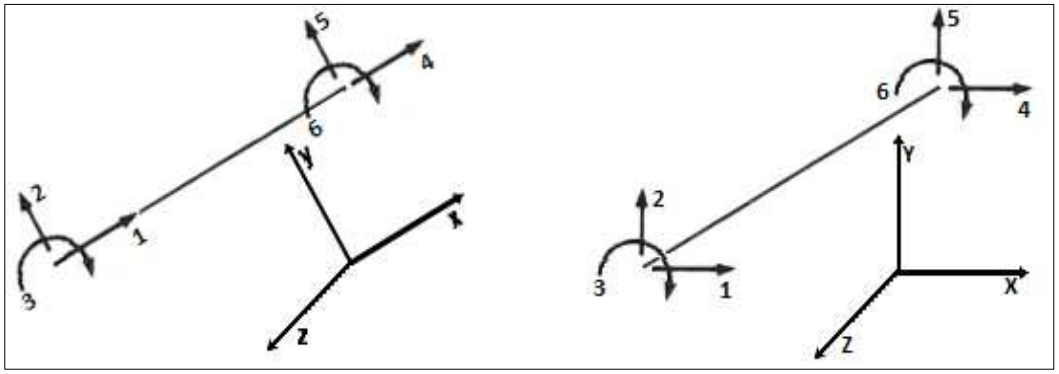
aplicadas à estrutura é descrito pelo vetor *F*, formado pelas componentes F_i, cada uma aplicada à coordenada i.

$$D = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \\ D_7 \\ D_8 \\ D_9 \\ D_{10} \\ D_{11} \\ D_{12} \end{bmatrix} \qquad \mathrm{F} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \\ F_{10} \\ F_{11} \\ F_{12} \end{bmatrix}$$

2.3.2 Coordenadas locais

Levando-se em consideração a barra de pórtico plano mostrada na Figura 03. As coordenadas locais podem ser definidas tanto no sistema global quanto no sistema local. Na figura, são ilustrados os eixos globais da estrutura (*XYZ*) e os eixos locais da barra (*xyz*).

Figura 03 – Coordenadas locais



Fonte: Martha (1993)

O vetor *d* define os deslocamentos das extremidades da barra no sistema de eixos globais, enquanto que o vetor *d'* define os deslocamentos no sistema de eixos locais. Os dois vetores são apresentados a seguir:

$$d = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} \qquad d' = \begin{bmatrix} d_1' \\ d_2' \\ d_3' \\ d_4' \\ d_5' \\ d_6' \end{bmatrix}$$

De forma similar, vetor *f* define as forças que atuam nas extremidades da barra segundo o sistema de eixos globais, enquanto que o vetor *f '* define as forças que atuam no sistema de eixos locais. Os dois vetores são mostrados a seguir:

$$f = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix} \qquad f' = \begin{bmatrix} f_1' \\ f_2' \\ f_3' \\ f_4' \\ f_5' \\ f_6' \end{bmatrix}$$

As forças, quando representadas no sistema de eixos locais, identificam o esforço normal, o esforço cortante e o momento fletor nas extremidades da barra.

2.4 Condições de equilíbrio

As estruturas devem ser capazes de alcançar um estado de equilíbrio estável para um determinado carregamento aplicado. Esta condição de equilíbrio deve ser satisfeita pela estrutura como um todo ou por qualquer porção isolada da estrutura (MARTHA, 1993).

Para uma estrutura bidimensional, as condições de equilíbrio resultam em três equações de equilíbrio, impondo-se que as resultantes de força e de momento sejam nulas:

$$\sum F_x = 0 \quad \sum F_y = 0 \quad \sum M_z = 0$$

O modelo matemático adotado neste trabalho utiliza a condições de equilíbrio aplicadas às coordenadas globais. Dessa forma, a força resultante aplicada na direção de cada coordenada global sempre será nula.

Desta forma, garante-se o equilíbrio dos pontos em que estão definidas as coordenadas globais, denominados de nós. Chegando-se ao equilíbrio de todos os nós da estrutura, alcança-se o equilíbrio da estrutura como um todo e de cada porção isolada dela.

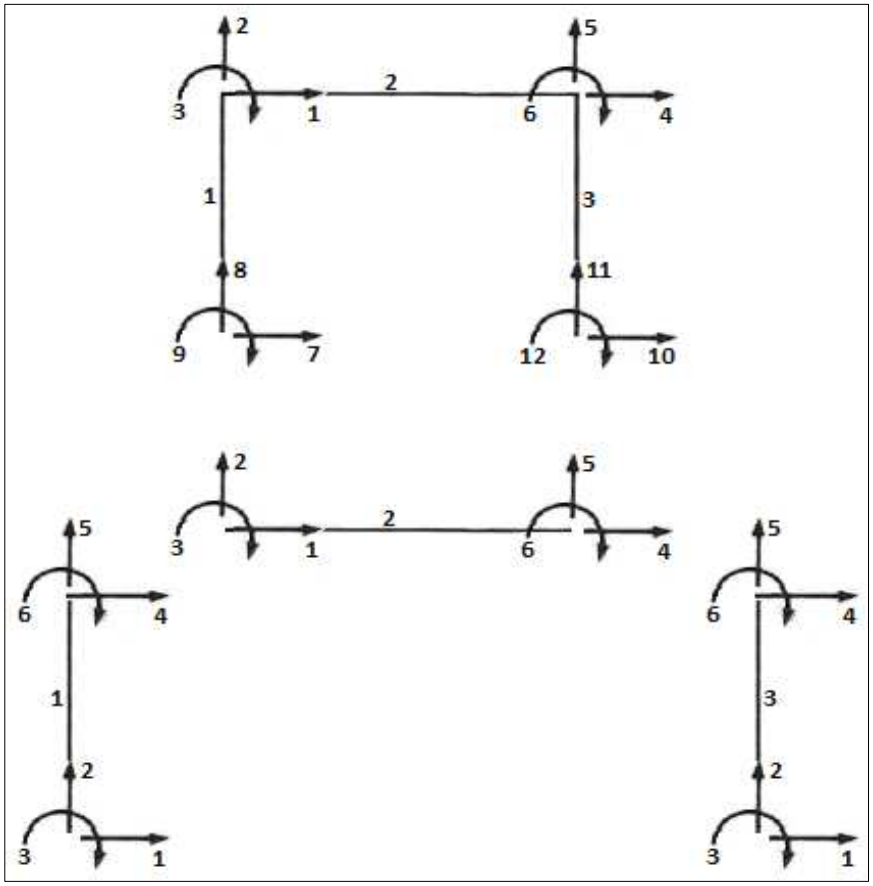
2.5 Condições de compatibilidade de deslocamentos

De acordo com Martha (1993), a compatibilidade de deslocamentos (e rotações) constitui um importante conceito da análise estrutural, que expressa a exigência de que todas as partes constituintes da estrutura deformada permaneçam ajustadas, unidas e ligadas durante todos os estágios do carregamento, significando que todos os deslocamentos e deformações dos vários elementos devem ser consistentes.

Em elementos estruturais constituídos por barras, a hipótese de deformação da barra mantendo a seção transversal plana, adotada neste trabalho, implica na existência de compatibilidade no interior delas. As relações entre deslocamentos e deformações, provenientes desta hipótese, para efeitos axiais e de flexão, serão mostrados na seção 2.5.1.

Assim, deve-se assegurar a compatibilidade nas junções das barras para que se possa garantir a compatibilidade no interior da estrutura. Para demonstrar as relações que garantem a compatibilidade nodal, a Figura 04 mostra as coordenadas globais e locais para a estrutura da Figura 01.

Figura 04 – Coordenadas globais e locais



Fonte: Martha (1993)

As condições de compatibilidade nas junções das barras são expressas por:

$$d_4^1 = d_1^2 = D_1$$
$$d_5^1 = d_2^2 = D_2$$
$$d_6^1 = d_3^2 = D_3$$
$$d_4^3 = d_4^2 = D_4$$
$$d_5^3 = d_5^2 = D_5$$
$$d_6^3 = d_6^2 = D_6$$

Em que d_j^i representa o efeito provocado pelo deslocamento *j* na barra *i*.

Tais condições, ditas de compatibilidade interna, partem do pressuposto de que as junções entre as barras são totalmente rígidas, ou seja, tanto deslocamentos quanto rotações são iguais nas duas extremidades da junta.

Além destas condições, os deslocamentos também devem respeitar as condições de apoio da estrutura, denominadas de compatibilidade externa. Para o caso da estrutura apresentada na Figura 04, as condições são as seguintes:

$$d_1^1 = D_7 = 0$$
$$d_2^1 = D_8 = 0$$
$$d_3^1 = D_9 = 0$$
$$d_1^3 = D_{10} = 0$$
$$d_2^3 = D_{11} = 0$$
$$d_3^3 = D_{12} = 0$$

2.5.1 Relações entre deslocamentos e deformações em barras

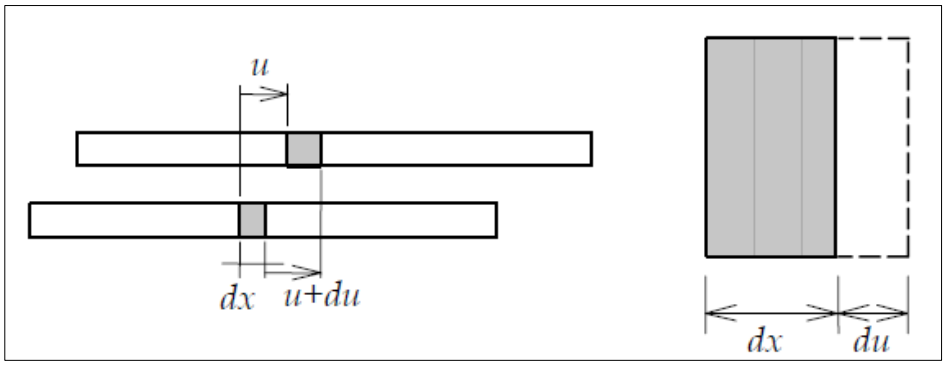
Segundo Marta (2010), o modelo estrutural tem como premissa uma condição de continuidade dos campos de deslocamentos e deformações no interior das barras, que devem ser compatíveis entre si, ou seja, deslocamentos e deformações devem estar relacionados.

2.5.1.1 Deformação axial

Denomina-se deformação axial, ε_x, a deformação normal à seção transversal provocada por efeitos axiais. Nesse tipo de deformação, todos os pontos de uma seção transversal têm sempre os mesmos deslocamentos axiais. Como consequência, verifica-se que

as seções transversais de uma barra submetida a uma deformação axial permanecem planas ao se deformarem, conforme ilustrado na Figura 05.

Figura 05 – Deslocamento axial de um elemento de barra infinitesimal.



Fonte: Martha (2010)

A deformação axial é igual à razão entre a variação de comprimento de um elemento infinitesimal e o seu comprimento inicial:

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

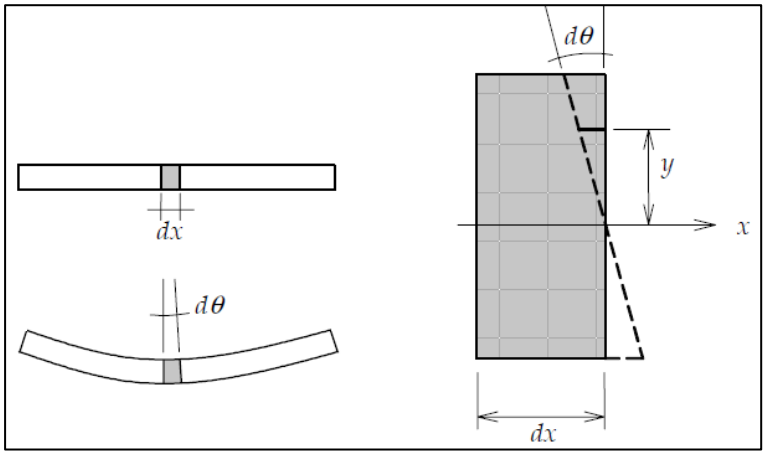
2.5.1.2 Deformações normais por flexão

Nesse tipo de deformação, considera-se que as seções transversais de uma barra que se deforma à flexão permanecem planas e normais ao eixo deformado da barra e que as deformações provocadas por efeitos de cisalhamento são desprezíveis. Estas condições garantem uma continuidade de deslocamentos no interior de uma barra que sofre flexão.

A deformação normal à seção transversal provocada por flexão é chamada de ε_f. A relação entre ε_f e o deslocamento *v* pode ser obtida pela análise da variação de comprimento de uma fibra genérica.

Cada fibra genérica do elemento infinitesimal é definida por uma coordenada *y*. Assim, considerando-se verdadeira a hipótese dos pequenos deslocamentos, tem-se que o encurtamento das fibras será de $d\theta \times y$, conforme mostrado na Figura 6.

Figura 06 – Rotação relativa de um elemento infinitesimal.



Fonte: Martha (2010)

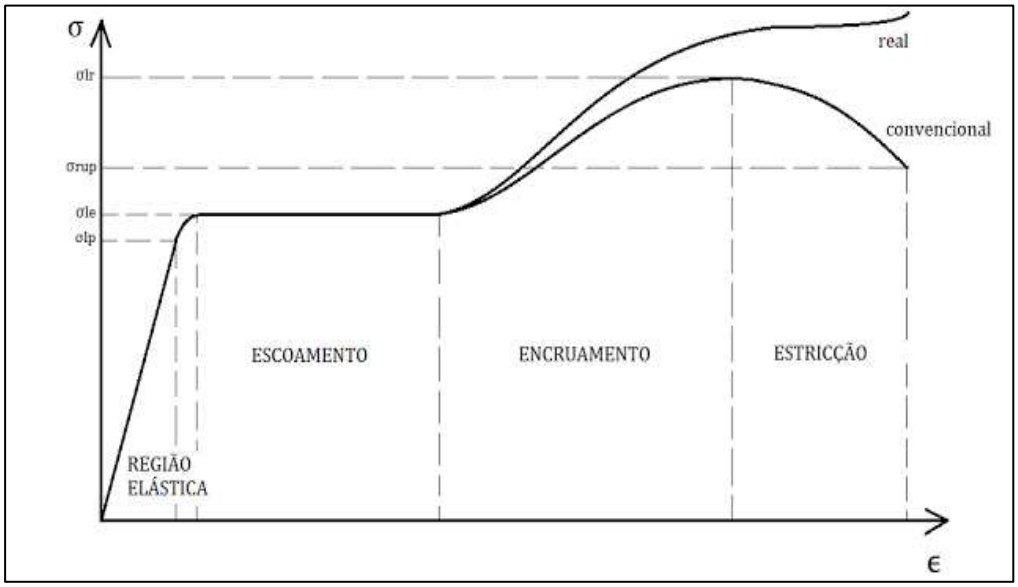
A deformação normal por flexão é igual a razão entre o encurtamento da fibra e o seu comprimento inicial, *dx*:

$$\varepsilon_f = -\frac{d\theta}{dx} y$$

2.6 A teoria da elasticidade linear

Todos os materiais possuem um limite de deformação que pode ser estudado por meio de um gráfico de tensão e deformação, como o mostrado na figura abaixo.

Figura 07 – Gráfico de tensão e deformação.



Fonte: Hibbeler (2010)

De acordo com Hibbeler (2010), cada região identificada neste diagrama apresenta carcterísticas singulares:

a) fase elástica: o material apresenta comportamento linear elástico, obedecendo à *lei de Hooke*, isto é, a tensão é proporcional à deformação ($\sigma = E \times \varepsilon_x$). O módulo de elasticidade (E), ou módulo de Young, é obtido por meio da inclinação da reta. A partir do limite de proporcionalidade (σ_{lp}), o material continua a apresentar comportamento elástico, contudo, a relação deixa de ser linear. Quando o material ultrapassa o chamado limite de escoamento (σ_{le}), a deformação se torna plástica, não retornando a sua forma original. A relação entre a tensão e a deformação pode ser expressa pela Lei de Hooke, na seguinte forma matricial:

$$\{\sigma_{ij}\} = [E]\{\varepsilon_{ij}\}$$

Em que:

$$\varepsilon_{xx} = \frac{du}{dx} \quad \varepsilon_{yy} = \frac{dv}{dy} \quad \varepsilon_{zz} = \frac{dw}{dz}$$

$$\{\varepsilon_{ij}\} = \{\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}\}$$

$$\{\sigma_{ij}\} = \{\sigma_{xx}, \sigma_{yy}, \sigma_{zz}\}$$

b) fase de escoamento: nessa etapa o material sofre uma brusca deformação, enquanto a tensão se mantém constante;

c) fase de encruamento: quando o material para de escoar, se uma carga adicional continua a ser aplicada, a curva cresce continuamente até atingir uma tensão limite, denominada limite de resistência (σ_{lr});

d) fase de estricção: a partir do limite de resistência, determinada região do material sofre uma diminuição na área da seção transversal. Com isso, a carga sofre um decréscimo até o momento em que o material se rompe. A falha do material ocorre na chamada tensão de ruptura (σ_{rup}).

2.7 Relações entre tensões e deformações

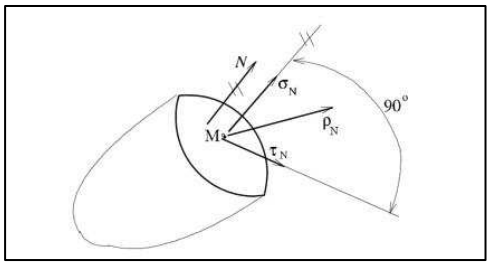
Segundo Hibbeler (2010), para o estudo das tensões no interior de um corpo deformado, deve-se isolar um elemento infinitesimal de dimensões dx, dy e dz. Este elemento infinitesimal, quando solicitado por forças externas, apresenta esforços em seu interior, que atuam na seção transversal produzindo tensões e gerando deformações.

Admitindo-se um plano passando pelo ponto *M* e que possui uma normal *N,* pode-se dizer que a tensão ρ_N, no ponto *M* no plano considerado, é a soma vetorial da tensão σ_N com tensão tangencial τ_N, conforme a Figura 08. Sua definição matemática é escrita como:

$$\rho N = \lim_{\Delta A \to 0} \frac{dF}{\Delta A}$$

em que *dF* é a força de interação atuante na área ΔA.

Figura 08 – Tensão no ponto M em um plano de normal N



Fonte: Hibbeler (2010)

Tomando-se então cada um dos três planos ortogonais *yz* (vetor normal paralelo ao eixo x), *xz* (vetor normal paralelo ao eixo y) e *xy* (vetor normal paralelo ao eixo z) é possível definir três vetores de tensões, respectivamente, que são fundamentais no estudo da grandeza tensão. Destaca-se que as tensões tangenciais totais são compostas por duas componentes:

$$\vec{\rho}_x = [\sigma_{xx}, \quad \tau_{xy}, \quad \tau_{xy}]$$
$$\vec{\rho}_y = [\tau_{yx}, \quad \sigma_{yy}, \quad \tau_{yz}]$$
$$\vec{\rho}_z = [\tau_{zx}, \quad \tau_{zy}, \quad \sigma_{zz}]$$

Os vetores $\vec{\rho}_x$, $\vec{\rho}_y$ e $\vec{\rho}_z$ são representados por meio do tensor de tensões que normalmente é simbolizado pela letra grega σ, conforme mostrado na equação:

$$\sigma = \begin{bmatrix} \vec{\rho}_x \\ \vec{\rho}_y \\ \vec{\rho}_z \end{bmatrix} = \begin{bmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_{zz} \end{bmatrix}$$

Para o caso de estruturas reticuladas planas, em que são considerados apenas os efeitos das tensões normais à seção transversal, pode-se escrever:

$$\sigma = E \times \varepsilon_x$$

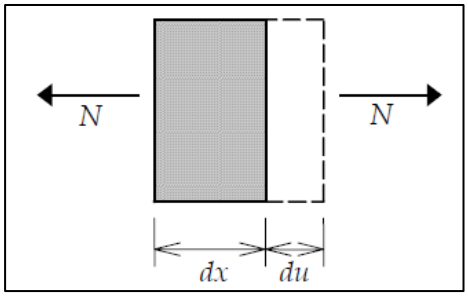
Essas relações de tensão-deformação são repassadas aos sistemas de coordenadas. Assim, servem de base para a determinação das relações entre forças e deslocamentos nos sistemas de coordenadas local e global.

2.7.1 Deslocamento relativo provocado por esforço normal

Para o efeito axial, tem-se que o deslocamento relativo provocado por um esforço normal atuando no elemento infinitesimal da Figura 09 é igual a:

$$N = \sigma_x \times A = E \times \varepsilon_x \times A = E \times \frac{du}{dx} \times A \longrightarrow du = \frac{N}{EA} dx$$

Figura 09 – Deslocamento relativo por esforço normal



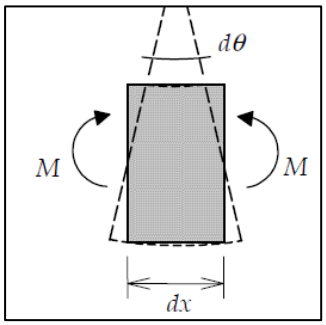
Fonte: Martha (2010)

2.7.2 Rotação relativa provocada por momento fletor

Para o efeito da flexão, tem-se que a rotação relativa provocada por um momento fletor atuando no elemento infinitesimal da Figura 10 é igual a:

$$M = \int_A \sigma_x (-y) dA = \int_A E \varepsilon_f (-y) dA = \int_A E \left(-\frac{d\theta}{dx} y\right)(-y) dA$$
$$= E \left(\frac{d\theta}{dx}\right) \int_A y^2 dA = EI \left(\frac{d\theta}{dx}\right) \longrightarrow d\theta = \frac{M}{EI} dx$$

Figura 10 – Rotação relativa por momento fletor



Fonte: Martha (2010)

2.8 Princípio da superposição de efeitos

Segundo Martha (1993), este princípio prescreve que todos os deslocamentos resultantes de um número de sistemas de forças podem ser somados para dar o deslocamento resultante da soma de todos os sistemas de forças. A superposição também implica o inverso: as forças correspondentes de um número de deslocamentos podem ser somadas para dar a força correspondente à soma dos deslocamentos.

De acordo com Beer et al. (2006), a superposição assegura que a tensão ou o deslocamento resultante em um ponto qualquer da estrutura pode ser determinado se antes for estabelecida a tensão ou o deslocamento causado por cada componente de carga agindo separadamente sobre o elemento.

Conforme explica Martha (1993), para que esse princípio possa ser utilizado, é necessário que a estrutura apresente um comportamento linear, isto é, os materiais devem ser elásticos e os deslocamentos devem ser pequenos. Nesse contexto, os deslocamentos podem ser considerados pequenos quando as geometrias inicial e final da peça são iguais em termos práticos.

2.9 Estruturas estaticamente determinadas e indeterminadas

Segundo Soriano (2004), as estruturas estáveis podem ser divididas em dois tipos: as estaticamente determinadas e as estaticamente indeterminadas, que são, respectivamente, denominadas de estruturas isostáticas e estruturas hiperestáticas.

2.9.1 Estruturas estaticamente determinadas

De acordo com Soriano (2004), são as estruturas que podem ter seus esforços internos e externos (reações de apoio) determinados, para qualquer caso de carregamento, apenas pelas condições de equilíbrio.

Essas estruturas possuem apenas o número de vínculos (internos ou externos) necessário para que permaneça estável. Retirando-se um desses vínculos, a estrutura torna-se instável (hipostática). Por outro lado, adicionando-se um vínculo, a estrutura torna-se hiperestática, pois esse vínculo não exerceria nenhuma influência no equilíbrio da estrutura.

2.9.2 Estruturas estaticamente indeterminadas

Consoante Soriano (2004), são as estruturas que não podem ter seus esforços internos e externos (reações de apoio) determinados, para qualquer caso de carregamento, apenas pelas condições de equilíbrio. Assim, além de utilizar as condições de equilíbrio, também é necessário fazer uso das condições de compatibilidade de deslocamentos para a determinação destes esforços.

Essas estruturas possuem mais vínculos (internos ou externos) do que os necessários para que ela seja estável. Por isso, retirando-se um destes vínculos, a estrutura continua estável.

Nesse tipo de estrutura, segundo Leet et al. (2010), as equações de equilíbrio não fornecem equações suficientes para a sua análise. Para tanto, deve-se derivar equações adicionais a partir da estrutura deformada, para complementar as equações de equilíbrio.

2.9.3 Comparação entre os tipos de estruturas

Em estruturas estaticamente determinadas (isostáticas), consegue-se determinar os esforços internos com base somente nos carregamentos. Por isso, para cada caso de carregamento, existe somente uma única solução para estes esforços. Além disso, pequenas variações na geometria de estruturas estaticamente determinadas não introduzem esforços adicionais nela, pois não chegam a alterar suas equações de equilíbrio.

Por outro lado, em estruturas estaticamente indeterminadas (hiperestáticas), existem inúmeras soluções que satisfazem às condições de equilíbrio para um mesmo carregamento. Portanto, a solução correta é aquela que satisfaz, além das condições de equilíbrio, a compatibilidade dos deslocamentos, o que torna mais complexa a análise desse tipo de estrutura.

De acordo com Martha (1993), apesar das vantagens oferecidas pelas estruturas isostáticas, as hiperestáticas são utilizadas na maioria das vezes. Para o autor, as razões que levam à escolha deste tipo de estrutura são:

a) a existência de formas estruturais intrinsicamente hiperestáticas, tais como edifícios e treliças espaciais;

b) a verificação de esforços menores que os normalmente encontrados em estruturas isostáticas correspondentes, por conta da melhor distribuição de esforços nas estruturas hiperestáticas;

c) o maior controle dos esforços internos por parte do projetista, que pode variar a rigidezes relativas das barras da estrutura para alterar os seus esforços internos. Isto não pode ser feito em estruturas isostáticas, pois existe uma única solução para os esforços internos; e

d) a segurança adicional introduzida por vínculos excedentes, que redistribuem os esforços quando parte da estrutura está sobrecarregada.

2.10 Métodos básicos da análise estrutural

De acordo com Martha (1993), a análise de estruturas consiste na determinação de uma solução que satisfaça as condições de equilíbrio, condições de compatibilidade de deslocamentos e relações de tensão-deformação. Os métodos básicos da análise estrutural se diferenciam pela ordem em que estas condições são impostas, conforme será apresentado nos tópicos seguintes.

2.10.1 Método das forças (ou da flexibilidade)

De acordo com Martha (2010), o método das forças resolve os problemas considerando os grupos de condições do modelo estrutural na seguinte ordem: primeiro as condições de equilíbrio; depois as condições sobre o comportamento dos materiais; e, por último, as condições de compatibilidade.

A metodologia utilizada na análise de uma estrutura hiperestática consiste no somatório de uma série de soluções básicas que satisfazem as condições de compatibilidade da estrutura original, para, por meio da superposição, reestabelecer as condições de compatibilidade.

Cada solução básica não satisfaz isoladamente todas as condições de compatibilidade da estrutura original, as quais ficam reestabelecidas quando se superpõem todos os casos básicos.

A estrutura utilizada para superposição de soluções básicas é, geralmente, uma estrutura isostática auxiliar, chamada de Sistema Principal (SP), obtida a partir da estrutura original pela eliminação de vínculos. As forças ou os momentos associados aos vínculos liberados são as incógnitas do problema e são denominados de hiperestáticos.

Como a estrutura é hiperestática, não é possível determinar os valores das reações de apoio da estrutura utilizando apenas as três equações de equilíbrio que estão disponíveis

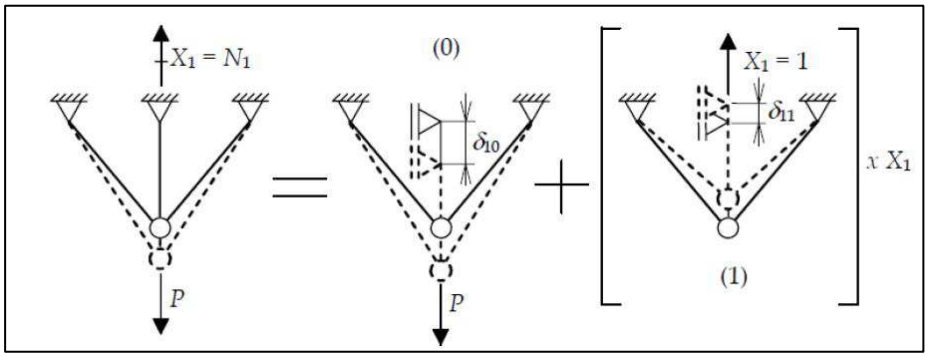
($\sum F_x = 0; \sum F_y = 0; \sum M_o = 0$). O número de incógnitas excedentes ao número de equações de equilíbrio é definido como *grau de hiperestaticidade* (g).

Conforme mencionado, a solução do problema hiperestático pelo presente método é feita por meio da superposição de soluções básicas isostáticas. Para isso, cria-se uma estrutura isostática auxiliar, chamada de Sistema Principal (SP), que é obtida da estrutura original hiperestática por meio da substituição de vínculos por redundantes estáticas X_i, conforme apresentado na Figura 11.

$$\delta_{10} + \delta_{11}X_1 + \delta_{12}X_2 + \cdots + \delta_{1N}X_N = 0$$
$$\delta_{20} + \delta_{21}X_1 + \delta_{22}X_2 + \cdots + \delta_{2N}X_N = 0$$
$$\vdots$$
$$\delta_{M0} + \delta_{M1}X_1 + \delta_{M2}X_2 + \cdots + \delta_{MN}X_N = 0$$

Em que δ_M representa os deslocamentos dos nós. Após a obtenção das referidas redundantes, utilizam-se as leis da estática para a determinação das reações e deslocamentos reais.

Figura 11 – Resolução pelo método das forças.



Fonte: Martha (2010)

2.10.2 Método dos deslocamentos

Segundo Martha (2010), o método dos deslocamentos pode ser considerado como o método dual do método das forças. Em ambos, a solução de uma estrutura considera os três grupos de condições básicas da análise estrutural: condições de equilíbrio, condições de compatibilidade entre deslocamentos e deformações e condições impostas pelas leis constitutivas dos materiais. Entretanto, o método dos deslocamentos resolve o problema considerando os grupos de condições do modelo estrutural na ordem inversa.

Conforme o autor, a dualidade dos métodos torna-se ainda mais evidente quando se observa a metodologia utilizada pelo método dos deslocamentos para analisar uma estrutura. A metodologia de cálculo consiste no somatório de uma série de soluções básicas que satisfazem as condições de compatibilidade, mas que não satisfazem as condições de equilíbrio da estrutura original, para, por meio de superposição, restabelecer as condições de equilíbrio. Portanto, realiza-se o procedimento inverso ao efetuado no método das forças.

De acordo com Martha (2010), cada caso básico satisfaz isoladamente as condições de compatibilidade (continuidade interna e compatibilidade com respeito aos vínculos externos da estrutura). No entanto, segundo o eminente autor, os casos básicos não satisfazem as condições de equilíbrio da estrutura original pois são necessários forças e momentos adicionais para manter o equilíbrio. Assim, as condições de equilíbrio da estrutura ficam restabelecidas quando se superpõem todas as soluções básicas.

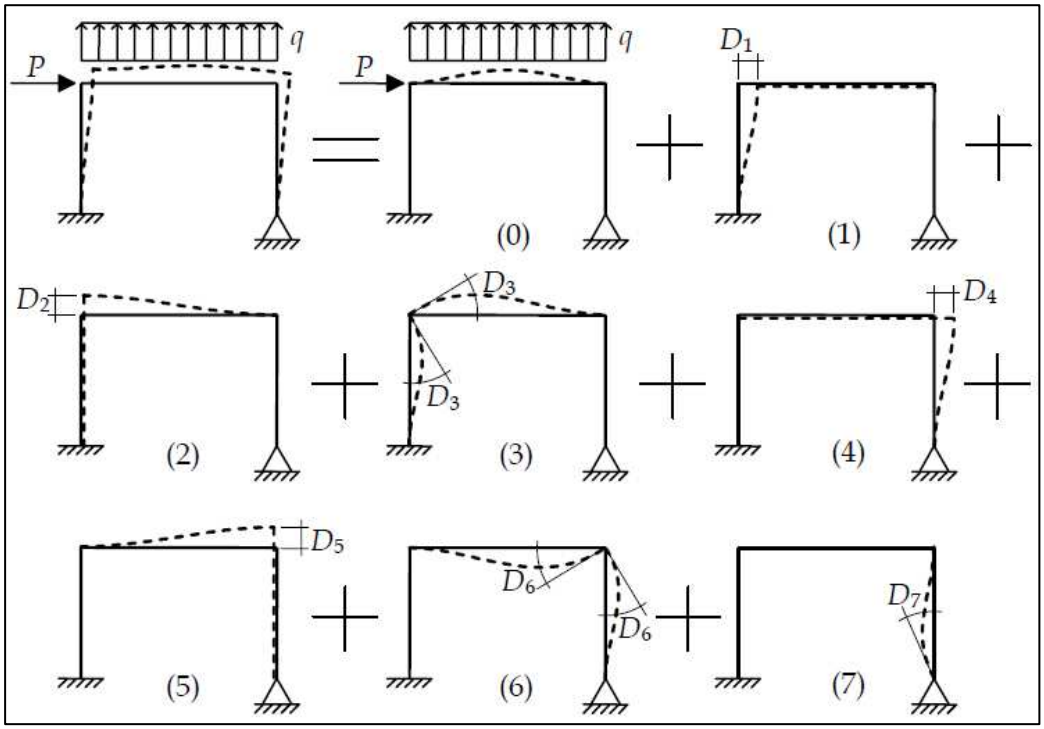
A configuração deformada da estrutura fica parametrizada pelas deslocabilidades (*Di*). Existem infinitas configurações deformadas que satisfazem as condições de compatibilidade com respeito aos vínculos externos (apoios), que respeitam as condições de continuidade do campo de deslocamentos no interior das barras e que atendem a continuidade de ligação entre as barras (as barras permanecem ligadas e com o mesmo ângulo entre si no nó interno). Entretanto, somente uma dessas configurações deformadas está associada ao equilíbrio da estrutura, conforme apresentado na Figura 12.

O método dos deslocamentos utiliza a estratégia de procurar, dentre todas as configurações deformadas que satisfazem a compatibilidade, aquela que também faz com que o equilíbrio seja satisfeito.

Consoante Süssekind (1987), o equilíbrio da estrutura é imposto na forma de equilíbrio dos nós isolados, considerando-se que as barras isoladas também estão em equilíbrio. Portanto, a solução desse problema pelo método dos deslocamentos recai em encontrar os valores que as deslocabilidades devem ter para que os nós internos fiquem em equilíbrio, pois os nós dos apoios têm seu equilíbrio automaticamente satisfeito pelas reações de apoio.

Segundo a abordagem do método dos deslocamentos, as soluções básicas isolam o efeito da solicitação externa e os efeitos de cada uma das deslocabilidades. Cada efeito isolado afeta o equilíbrio do nó interno. Na superposição dos casos básicos é imposto o equilíbrio do nó interno. Os casos básicos utilizam um sistema hipergeométrico (SH) como estrutura auxiliar, através da qual os efeitos isolados são impostos.

Figura 12 – Configuração deformada da estrutura.



Fonte: Martha (2010)

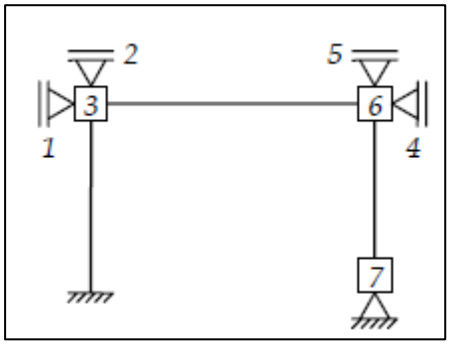
2.10.2.1 O sistema hipergeométrico

O sistema hipergeométrico é uma estrutura fictícia que mantém as características da estrutura inicial, mas tem todas as suas juntas completamente engastadas, para isto estabelecendo apoios fictícios. Os engates permitem que cada uma das deslocabilidades (*Di*) da estrutura seja isolada, sem que uma exerça influência nas demais. Por não possuir juntas livres, suas deslocabilidades não se originam da aplicação de cargas, mas sim do recalque de apoios fictícios.

Na Figura 13 foram adicionados os apoios fictícios à estrutura para que esta se adequasse ao modelo hipergeométrico. A cada um deles foi atribuído um índice, que corresponde também ao índice da deslocabilidade gerada pelo seu recalque. Dessa forma, quando o apoio *n* for recalcado, notar-se-á o surgimento da deslocabilidade *Dn*.

Para dar continuidade à resolução da estrutura, deve-se definir como serão calculados os casos básicos nos quais a estrutura foi decomposta. Levando-se em consideração o pórtico apresentado, divide-se o cálculo em duas etapas: na primeira, computam-se os esforços causados pelo carregamento da estrutura, isto é, o caso básico (0). Depois, calculam-se os demais casos, com o auxílio da metodologia apresentada a seguir.

Figura 13 – O sistema hipergeométrico.



Fonte: Martha (2010)

2.10.2.2 Resolução do caso básico (0)

Ao se analisar as cargas aplicadas ao sistema hipergeométrico do caso básico (0), percebe-se que este sistema pode ser decomposto em vários elementos diferentes, que podem ser calculados individualmente. Isso ocorre porque, neste caso, não existem transferências de esforços de um elemento para os demais, pois todas as juntas encontram-se engastadas. Assim, para determinar as reações da estrutura, basta determinar as reações em cada um dos elementos que a compõem.

Estes esforços são denominados de forças de engastamento perfeito. Neste trabalho, optou-se por utilizar a notação F_{i0} para representar esses esforços, seguindo a mesma abordagem utilizada para denotar as deslocabilidades, o que significa que o valor F_{i0} está relacionado à reação no apoio associado com D_i.

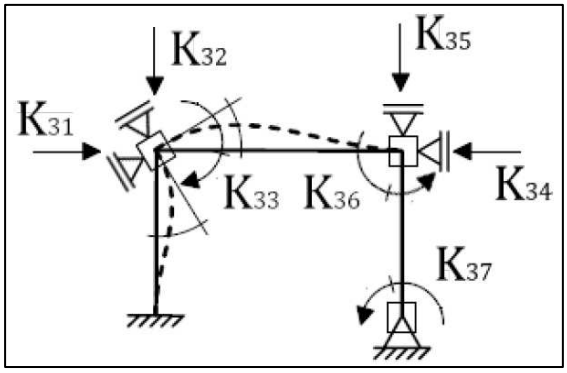
2.10.2.3 Resolução dos demais casos básicos

De acordo com Martha (2010), diferentemente do que ocorre no caso básico (0), nos demais casos, não serão as cargas que causarão as reações nos apoios fictícios, mas sim o recalque de qualquer um destes apoios. Para esses casos, deve-se admitir um deslocamento unitário nos apoios fictícios, de modo que os efeitos causados por este recalque sejam multiplicados pelo valor de *Di*.

Segundo Longo (2015), os coeficientes de rigidez global da estrutura indicam a capacidade que a estrutura tem de se manter estável perante os carregamentos que lhes são impostos. Assim, quanto maiores forem estes valores, menores serão os deslocamentos identificados em cada um de seus pontos. Estes coeficientes são representados por *Kij*, em que *i* se refere ao apoio fictício que foi deslocado e *j* o apoio fictício em que foram verificados os

efeitos. Para o pórtico da Figura 14, o índice j variará de 1 a 7, sendo ($K_{31}, K_{32}, ..., K_{37}$) os seus respectivos coeficientes de rigidez.

Figura 14 – Coeficientes de rigidez global.



Fonte: Longo (2015)

2.10.2.4 Rótulas internas

As articulações internas são ligações entre elementos de uma determinada estrutura que permitem algum tipo de deslocamento entre eles, tendo, por característica, não contribuir com a rigidez associada à deslocabilidade sobre a qual agem. O único tipo de articulação abordado neste trabalho é a rótula, que possibilita a existência de rotações relativas entre as seções transversais de barras.

Segundo Martha (2010), a utilização de rótulas assegura a liberação da continuidade da rotação, permitindo que dois elementos adjacentes apresentem rotações diferentes. Além disso, as rótulas não transmitem momentos aos elementos vizinhos, efetuando apenas a transmissão de forças. Como consequência, caso não haja uma carga fletora aplicada ao nó rotulado, o momento fletor deste ponto será igual a zero.

De acordo com Kassimali (2012), a utilização de articulações é capaz de modificar a distribuição de esforços de uma determinada estrutura e, por conta disso, esta condição deve ser levada em consideração na análise do modelo estrutural. Estes elementos podem ser encontrados de duas maneiras nas estruturas: associados às extremidades das barras ou vinculados a um determinado nó, circunstância em que todas as barras ligadas a esse nó são consideradas rotuladas em sua extremidade.

Dessa forma, pode-se identificar quatro situações de rotulação para cada uma das barras: duas extremidades rígidas, uma das extremidades rotuladas (contabilizando-se duas possibilidades) ou duas extremidades rotuladas.

2.10.2.5 Compatibilidade estática

Tendo determinado os efeitos causados pelas cargas em cada um dos apoios e de posse dos coeficientes de rigidez global da estrutura, faz-se necessário realizar a compatibilização estática da estrutura

Pode-se escrever o sistema de equações de equilíbrio para *n* deslocabilidades da seguinte maneira:

$$\begin{cases} F_{10} + K_{11}D_1 + K_{12}D_2 + \cdots + K_{1n}D_n = 0 \\ F_{20} + K_{21}D_1 + K_{22}D_2 + \cdots + K_{2n}D_n = 0 \\ \vdots \\ F_{n0} + K_{n1}D_1 + K_{n2}D_2 + \cdots + K_{nn}D_n = 0 \end{cases}$$

Esse sistema de equações possibilita a determinação dos valores de D_i, que, posteriormente, permitirão determinar as reações da estrutura em seus apoios. Por se tratar de um sistema de equações, pode-se escrevê-lo em sua forma matricial, no seguinte formato:

$$\begin{Bmatrix} F_{10} \\ F_{20} \\ \vdots \\ F_{n0} \end{Bmatrix} + \begin{bmatrix} K_{11} & K_{12} & \dots & K_{13} \\ K_{21} & K_{22} & \dots & K_{23} \\ \vdots & \vdots & \ddots & \vdots \\ K_{n1} & K_{n2} & \dots & K_{nn} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ \vdots \\ D_n \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}$$

No caso de uma estrutura com *n* deslocabilidades, representa-se o sistema no seguinte formato reduzido:

$$\{F\} + [K]\{D\} = \{0\}$$

Sendo:

$\{F\}$: vetor global das forças de engastamento perfeito;

$[K]$: matriz de rigidez global da estrutura; e

$\{D\}$: vetor global de deslocamentos da estrutura.

2.11 Métodos baseados em energia

Assim como os métodos baseados em relações de tensão e deformação, os métodos baseados em energia também permitem determinar os deslocamentos em estruturas, e faz isso, normalmente, de um modo mais conveniente. Ademais, os métodos de energia podem ser empregados para desenvolver equações adicionais, que podem auxiliar na resolução de estruturas estaticamente indeterminadas, facilitando a análise de forças e deslocamentos desconhecidos.

2.11.1 Princípio da conservação da energia

Segundo Hibbeler (2013), as deformações surgem quando um carregamento é aplicado a um corpo. Caso nenhuma energia seja perdida sob forma de calor, o trabalho externo realizado pelas cargas será convertido em trabalho interno denominado energia de deformação. Essa energia é armazenada no corpo e provocada pela ação da tensão normal ou tensão de cisalhamento.

Conforme o referido autor, se o carregamento for aplicado lentamente, de modo que a energia cinética também possa ser desprezada, as cargas externas tendem a deformar o corpo fisicamente, de modo que as cargas realizam um trabalho externo U_e à medida que são deslocadas. Esse trabalho externo provocado pelas cargas transforma-se em trabalho interno, ou energia de deformação U_i, que é armazenada no corpo.

Quando as cargas são removidas, desde que o limite de elasticidade do material não tenha sido ultrapassado, a energia de deformação é capaz de restituir o corpo à posição original não deformada.

Dessa forma, o princípio da conservação de energia para o corpo pode ser expresso matematicamente como:

$$U_e = U_i$$

2.11.2 Teorema de Claperyon para a elasticidade

Segundo o Teorema de Clapeyron para a elasticidade, a energia potencial de deformação de uma estrutura que se encontra em equilíbrio para um determinado carregamento é equivalente à metade do trabalho realizado pelas forças externas, atuando por meio de deslocamentos desde o estado inicial até o estado de equilíbrio.

Dessa forma, sejam $P_1, P_i, \dots, P_n$ forças externas independentes e $\delta_1, \delta_i, \dots, \delta_n$ os deslocamentos correspondentes de seus pontos de aplicação, medidos na direção e no sentido de cada uma das forças. Admitindo-se que as forças P_i possam ser aplicadas gradualmente e que, em determinado instante, as forças assumem a forma αP_i, com α variando entre 0 e 1, tem-se que os deslocamentos também podem ser colocados na sob a forma $\alpha\delta_i$, pois obedecem à Lei de Hooke.

Com a passagem de um estado de solicitação para outro infinitamente próximo, isto é, com o incremento mínimo de α, o deslocamento genérico $\alpha.\delta_i$ torna-se $(\alpha + d\alpha).\delta_i$, promovendo um acréscimo de trabalho igual a:

$$dU = \sum_{i=1}^{n} \alpha P_i . d\alpha \, \delta_i = \sum_{i=1}^{n} P_i \delta_i \alpha \, d\alpha$$

O trabalho total realizado por todas as forças é:

$$U = \sum_{i=1}^{n} \int_{\alpha=0}^{1} \alpha P_i . d\alpha \, \delta_i = \frac{1}{2} \sum_{i=1}^{n} P_i \delta_i$$

Adotando-se esta mesma abordagem, pode-se deduzir a expressão da energia total para um carregamento de momentos, que é equivalente à:

$$U = \frac{1}{2} \sum_{i=1}^{n} M_i \phi_i$$

2.11.3 Teorema de Castigliano

O Teorema de Castigliano permite determinar o deslocamento e a rotação associados a um ponto de um corpo. Conforme o teorema, a derivada parcial da energia de deformação de um sistema, com relação a uma força, é igual ao deslocamento do ponto de aplicação da força na direção e no sentido da força. De acordo com Hibbeler (2013), o método somente pode ser aplicado aos materiais que se comportam de modo elástico-linear.

Seja um corpo elástico, com energia de deformação *U*, solicitado por um sistema de forças externas e apoiado de forma a não permitir o seu movimento. Supondo-se que seja aplicado um incremento dP_i à força externa P_i, o aumento na energia de deformação será de:

$$dU = \frac{\partial U}{\partial P_i} \times dP_i$$

em que $\partial U / \partial P_i$ é a taxa de variação de *U* com relação a P_i. A energia de deformação da barra é equivalente a:

$$U + du = U + \frac{\partial U}{\partial P_i} \times dP_i$$

Visto que o princípio da superposição é válido para esta barra, a energia de deformação é independente da ordem em que os carregamentos são aplicados. Assim, quando o carregamento dP_i é aplicado primeiro, ele produz energia de deformação igual à metade do produto do carregamento dP_i pelo seu deslocamento $d\delta_i$:

$$\frac{dP_i \times d\delta_i}{2}$$

Aplicando-se ao restante do sistema de forças externas, pode-se concluir que a energia de deformação final para o carregamento é:

$$U + \frac{dP_i \times d\delta_i}{2} + dP_i \times \delta_i$$

Pelo princípio da conservação de energia, tem-se:

$$U + \frac{\partial U}{\partial P_i} \times dP_i = U + \frac{dP_i \times d\delta_i}{2} + dP_i \times \delta_i$$

O segundo termo da equação pode ser descartado, pois resulta do produto de dois diferenciais e apresenta um valor muito pequeno se comparado aos demais termos. Assim, a equação se reduz a:

$$\frac{\partial U}{\partial P_i} = \delta_i$$

Portanto, o deslocamento devido a um esforço axial pode ser obtido pela seguinte expressão:

$$\frac{\partial}{\partial P}\left(\frac{N^2 L}{2EA}\right) = \delta$$

Reescrevendo-a de forma a isolar deslocamento e esforço, obtém-se:

$$N = \frac{EA}{L}\delta$$

2.11.4 Funcional de energia potencial

Quando um corpo elástico está em equilíbrio, verifica-se a condição de energia potencial mínima. A energia potencial pode ser definida como a soma da energia de deformação *U* com o trabalho realizado pelas forças externas *W*, isto é:

$$\Pi_P = U + W$$

em que Π_P representa o funcional de energia potencial.

Para aplicar o princípio da energia potencial mínima, pode-se calcular a variação de π_P e igualar a zero. Logo, assumindo que o carregamento permanece constante, tem-se que:

$$\delta\Pi_P = \delta U + \delta W = 0$$

A variação pode ser interpretada como a composição das derivadas parciais de π_P em relação às coordenadas independentes, ou variáveis em termos das quais está expressa:

$$\Pi_P = \Pi_P(u_{x1}, u_{x2}, \dots, u_{x2})$$

em que $(u_{x1}, u_{x2}, \dots, u_{x2})$ representam o número total de variáveis dos nós.

Dessa forma, tem-se que:

$$\delta \Pi_P = 0$$

Implicando que:

$$\frac{\partial \Pi_P}{\partial u_{x1}} = 0 \quad \frac{\partial \Pi_P}{\partial u_{x2}} = 0 \quad \ldots \quad \frac{\partial \Pi_P}{\partial u_{xn}} = 0$$

2.11.5 Princípio dos trabalhos virtuais

De acordo com Süssekind (1980), pelo princípio do trabalho virtual aplicado aos corpos elásticos, o trabalho virtual das forças externas é igual ao trabalho virtual das forças internas em quaisquer deslocamentos virtuais compatíveis com os vínculos da estrutura.

$$\delta W_e = \delta W_i$$

O estado de carregamento escolhido deve ser tal que a carga virtual *P* associada ao deslocamento δ (que se pretende calcular) forneça um trabalho virtual de forças externas igual a $P.\delta$.

Por esse princípio, primeiramente deve ser aplicada uma força virtual *P* na direção do deslocamento que se deseja calcular. A aplicação desta força resultará no surgimento de esforços internos virtuais de flexão ($\bar{M}$), normais ($\bar{N}$), de cisalhamento ($\bar{V}$) e de torção ($\bar{T}$). Após isso, com a força virtual atuando na estrutura, pode-se aplicar as forças reais ou induzir deformações específicas.

Para Süssekind (1980), o trabalho realizado pela força virtual *P* deslocando-se δ em sua direção é igual ao trabalho total realizado nos elementos internos pelas forças virtuais ($\bar{M}$, $\bar{N}$, $\bar{V}$, $\bar{T}$). O trabalho realizado pela força virtual está associado às deformações devidas à flexão, ao esforço normal, ao esforço cortante e à torção:

$$W_{int} = \int_l \bar{M} d\varphi + \int_l \bar{N} \Delta + \int_l \bar{Q} d\gamma + \int_l \bar{T} d\theta$$

Em que $d\varphi$ corresponde às rotações devidas ao carregamento real; Δ corresponde aos alongamentos ou encurtamentos deidos ao carregamento real; $d\gamma$ corresponde às deformações angulares e $d\theta$ corresponde às torções.

Dessa forma, tem-se que:

$$\bar{P}.\delta = \int_l \bar{M} \frac{M}{EI} dx + \int_l \bar{N} \frac{N}{EA} dx + \int_l \chi \bar{V} \frac{V}{GA} dx + \int_l \bar{T} \frac{T}{GJ_t} dx$$

Em que $\bar{M}$, $\bar{N}$, $\bar{V}$, $\bar{T}$ são os esforços produzidos pela carga virtual; *M, N, V* e *T* são os esforços produzidos pelo carregamento real; *E* é o módulo de elasticidade longitudinal; *G* é

o módulo de elasticidade transversal, I é o momento de inércia da seção transversal; A é a área da seção transversal e J_t é o momento de inércia à torção.

2.12 O método dos elementos finitos

Segundo Ribeiro (2004), o método dos elementos finitos consiste em subdividir o domínio da estrutura analisada em subdomínios de dimensões finitas, de forma que o conjunto de todos os subdomínios deve ser igual ao domínio original do problema. Em seguida, para cada subdomínio, adota-se um comportamento aproximado (local) para as incógnitas do problema.

Conforme explica o indigitado autor, o método implica na divisão do domínio da estrutura que se pretende analisar em subdomínios, chamados elementos finitos, que são interligados entre si por elementos nodais. Assim, as soluções do problema são formuladas para cada elemento e, então, são combinadas para obter a solução do domínio completo.

O método pode auxiliar na resolução de diversos problemas da Engenharia Civil, onde encontra aplicação nos campos da Mecânica dos Solos, Mecânica dos Fluidos e Mecânica das Estruturas. No âmbito da Engenharia de Estruturas, o método tem como objetivo a determinação do estado de tensão e de deformação de sólidos de geometria arbitrária sujeitos a ações exteriores.

Neste trabalho, a formulação do método dos elementos finitos está baseada no método dos deslocamentos, pois apresenta maior simplicidade em relação às demais abordagens. Além disso, está fundamentado no princípio da conservação de energia, no princípio do trabalho virtual e nas relações de tensão e deformação, mostrados nos tópicos anteriores.

2.12.1 A abordagem computacional

Segundo Azevedo (2003), a determinação das variáveis envolvidas no método dos deslocamentos não requer grande esforço de cálculo quando a geometria da estrutura é relativamente simples. No entanto, quando a geometria se torna mais complexa, envolvendo elementos inclinados, de seções variáveis ou materiais diferentes, o cômputo dos coeficientes de rigidez e das forças de engastamento perfeito revela-se muito mais dificultoso, podendo demandar, para tanto, o uso de soluções informatizadas.

Por isso, faz-se necessário segmentar a estrutura, dividindo seus elementos de modo que cada um seja analisado separadamente, sendo suas contribuições adicionadas posteriormente ao vetor de forças de engastamento perfeito e à matriz de rigidez global. Esta segmentação, de acordo com Kassimali (2012), requer a definição de um sistema de coordenadas locais para cada elemento da estrutura, que permitirá expressar as relações básicas de elasticidade dos membros em termos das forças e dos deslocamentos da estrutura.

Dessa forma, deve-se adotar dois sistemas de coordenadas: um posicionado em relação à estrutura como um todo, denominado de sistema global, e o outro posicionado em relação a cada um dos elementos, cuja origem localiza-se em seu nó inicial, designado de sistema local.

Segundo Longo (2015), o sistema local possibilita a definição de rotinas de cálculo mais sistemáticas, visto que trabalha examinando um elemento de cada vez, o que permite a realização de análises mais acuradas das relações de força e deslocamento. Conhecendo-se as propriedades de cada elemento, pode-se utilizar as relações de correspondência entre os sistemas global e local para incorporar as contribuições de cada elemento à equação global da estrutura.

Para isto, supõe-se a estrutura em sua forma não restringida, isto é, sem seus os apoios. Nessa situação, devem ser verificadas todas as possíveis deslocabilidades dos nós da estrutura. Após isso, volta-se a considerar a estrutura restringida, de forma que as equações referentes aos deslocamentos restritos podem ser eliminadas. Isto simplifica o estudo da estrutura, pois, quando as barras são analisadas isoladamente, não se faz necessário levar em conta as condições de apoio da estrutura.

2.12.2 Forças de engastamento perfeito

Para o cálculo das forças de engastamento perfeito, supõe-se que as barras são homogêneas e prismáticas, isto é, não ocorrem mudanças de material ou variação de seção ao longo do comprimento delas. Além disso, considera-se que as cargas podem estar submetidas a quatro condições de articulação, apresentando diferentes valores de forças de engastamento perfeito para cada uma delas, conforme mostrado na Figura 15.

Ademais, ressalta-se que, neste trabalho, as cargas distribuídas estão limitadas aos casos linearmente variáveis, podendo ser descritas por meio de um vetor de componentes x e y. Por sua vez, as cargas concentradas devem ser aplicadas exclusivamente aos nós da estrutura e são representadas por um vetor que será descrito nas próximas seções.

As forças de engastamento perfeito podem ser descritas por meio do vetor de forças, $\{f\}$, conforme apresentado a seguir:

$$\{f\} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{Bmatrix} = \begin{Bmatrix} H_0 \\ V_0 \\ M_0 \\ H_1 \\ V_1 \\ M_1 \end{Bmatrix}$$

Figura 15 – Forças de engastamento perfeito para os diferentes tipos de barras.

Tipo de Apoio	 0	 1	 2	 3
H_0	$-\frac{1}{6}\left[2q_{x0} + q_{x1}\right]L$			
H_1	$-\frac{1}{6}\left[q_{x0} + 2q_{x1}\right]L$			
V_0	$-\frac{1}{20}\left(7q_{y0} + 3q_{y1}\right)L$	$-\frac{1}{120}\left(33q_{y0} + 12q_{y1}\right)L$	$-\frac{1}{120}\left(48q_{y0} + 27q_{y1}\right)L$	$-\frac{1}{6}\left[2q_{y0} + q_{y1}\right]L$
V_1	$-\frac{1}{20}\left(3q_{y0} + 7q_{y1}\right)L$	$-\frac{1}{120}\left(27q_{y0} + 48q_{y1}\right)L$	$-\frac{1}{120}\left(12q_{y0} + 33q_{y1}\right)L$	$-\frac{1}{6}\left[q_{y0} + 2q_{y1}\right]L$
M_0	$-\frac{1}{120}\left(6q_{y0} + 4q_{y1}\right)$	0	$-\frac{1}{120}\left(8q_{y0} + 7q_{y1}\right)L^2$	0
M_1	$\frac{1}{120}\left(4q_{y0} + 6q_{y1}\right)$	$\frac{1}{120}\left(7q_{y0} + 8q_{y1}\right)L^2$	0	0

Fonte: Longo (2015)

2.12.3 Matriz de rigidez local

Supondo-se uma estrutura composta exclusivamente por elementos homogêneos e prismáticos, que podem estar submetidos a uma das quatro condições de apoio mostradas na Figura 15, os coeficientes locais, denominados de k_{ij}, serão determinados conforme estabelecido na seção 2.9.2.3, isto é, uma deslocabilidade unitária é imposta à barra e serve de base para o cômputo dos coeficientes de rigidez.

Em Kassimali (2012), encontram-se disponíveis as expressões para todas as configurações de rótulas possíveis:

a) articulada-articulada:

$$[K] = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

b) articulada-engastada:

$$[K] = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{3EI}{L^3} & 0 & 0 & -\frac{3EI}{L^3} & \frac{3EI}{L^2} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{3EI}{L^3} & 0 & 0 & \frac{3EI}{L^3} & -\frac{3EI}{L^2} \\ 0 & \frac{3EI}{L^2} & 0 & 0 & -\frac{3EI}{L^2} & -\frac{3EI}{L} \end{bmatrix}$$

c) engastada-articulada:

$$[K] = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{3EI}{L^3} & \frac{3EI}{L^2} & 0 & -\frac{3EI}{L^3} & 0 \\ 0 & \frac{3EI}{L^2} & \frac{3EI}{L} & 0 & -\frac{3EI}{L^2} & 0 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{3EI}{L^3} & -\frac{3EI}{L^2} & 0 & \frac{3EI}{L^3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

d) engastada-engastada:

$$[K] = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix}$$

2.12.4 Vetor de cargas pontuais

Visto que as cargas pontuais da estrutura serão aplicadas exclusivamente por meio dos nós, nunca por meio das barras, pode-se desmembrar o vetor de forças de engastamento perfeito, criando um vetor de cargas pontuais.

De acordo com Longo (2015), esta abordagem apresenta duas vantagens do ponto de vista computacional. Em primeiro lugar, permite que os casos de carregamento fiquem limitados às cargas linearmente distribuídas, cujos esforços podem ser calculados conforme mostrado na seção 2.11.2. Caso existam cargas pontuais aplicadas aos elementos, deve-se considerar os efeitos de cada uma delas individualmente, computando seus esforços separadamente e depois combinando-os para obter o esforço final, que, então, será repassado ao sistema global de coordenadas, por meio das regras de correspondência.

A outra vantagem é que basta determinar a que deslocabilidade da estrutura não restringida a carga atuante está associada e montar o vetor de ações nodais {P} diretamente no sistema global, de acordo com seus índices:

$$\{P\} = \begin{Bmatrix} P_0 \\ P_1 \\ P_2 \\ \vdots \\ P_{n-2} \\ P_{n-1} \\ P_n \end{Bmatrix}$$

2.12.5 Vetor de reações de apoio

O vetor {R} conterá os valores das reações de apoio da estrutura restringida, possuindo um elemento para cada uma de suas deslocabilidades, sendo atribuído o valor de zero aos nós que não possuírem restrição de deslocamento:

$$\{R\} = \begin{Bmatrix} R_0 \\ R_1 \\ R_2 \\ \vdots \\ R_{n-2} \\ R_{n-1} \\ R_n \end{Bmatrix}$$

2.12.6 Relações entre o sistema de coordenadas global e local

Uma vez que foram estabelecidos dois sistemas de coordenadas distintos para a implementação do método de cálculo, faz-se necessária a definição de equações que expressem as relações entre esses dois sistemas.

Estas relações levarão em consideração todos os aspectos do posicionamento das barras da estrutura, fazendo a correspondência entre os nós da barra no sistema local e os nós da estrutura no sistema global.

2.12.7 Matriz de rotação

Segundo Kassimali (2012), é necessário transformar as relações de rigidez das barras de um pórtico dos seus sistemas de coordenadas locais para o sistema de coordenada global, antes que possam ser combinadas para estabelecer as relações de rigidez da estrutura como um todo.

Assim, deve-se considerar a rotação dos elementos em relação ao sistema de coordenadas globais para garantir que suas parcelas de contribuição de rigidez, e também suas forças de engastamento perfeito, sejam consideradas nos eixos apropriados.

Para tanto, Martha (2010) definiu uma matriz de rotação $[T_\theta]$ para fazer as transformações entre os sistemas de coordenadas. Esta matriz depende do ângulo θ formado entre a barra e o eixo global horizontal da estrutura.

$$[T_\theta] = \begin{bmatrix} cos(\theta) & sen(\theta) & 0 & 0 & 0 & 0 \\ -sen(\theta) & cos(\theta) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & cos(\theta) & sen(\theta) & 0 \\ 0 & 0 & 0 & -sen(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

O autor Kassimali (2012) estabeleceu as equações para fazer estas transformações, de modo a se obter, no sistema de coordenadas global, para cada uma das barras, o seu vetor de engastamento perfeito $\{f_g\}$ e a sua matriz de rigidez $[K]$:

$$\{f_g\} = [T_\theta]^t \{f\}$$

$$[K_g] = [T_\theta]^t [K][T_\theta]$$

2.12.8 Matriz de correspondência

Segundo Longo (2015), após terem sido determinadas as variáveis da barra em termos de coordenadas globais, pode-se combinar os coeficientes de cada elemento na matriz de rigidez global. Para tanto, faz-se necessário definir o arranjo dos coeficientes de rigidez e das forças de engastamento perfeito nas suas matrizes e vetores correspondentes, assim como estabelecer um sistema de numeração para os elementos da estrutura.

Assim, deve-se ressaltar que tanto os nós quanto as barras da estrutura possuirão um índice numérico crescente e contado a partir de zero. Além disso, as barras também serão identificadas pelo índice dos seus nós. A ordem em que os nós forem informados indicará a orientação da barra e, assim, o seu ângulo em relação ao eixo horizontal.

Considerando-se que cada nó de um pórtico plano tenha três deslocabilidades, pode-se escrever o vetor de forças de engastamento perfeito da seguinte forma:

$$\begin{matrix} \boldsymbol{N_0} \\ \\ \\ \\ \boldsymbol{N_n} \end{matrix} \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-2} \\ f_{n-1} \\ f_n \end{Bmatrix}$$

Fazendo uso dessa mesma lógica, chega-se à seguinte matriz de rigidez para uma determinada estrutura:

$$\begin{matrix} \\ \boldsymbol{N_0} \\ \\ \\ \\ \boldsymbol{N_n} \\ \\ \end{matrix} \begin{bmatrix} K_{0,0} & K_{0,1} & K_{0,2} & & K_{0,n-2} & K_{0,n-1} & K_{0,n} \\ K_{1,0} & K_{1,1} & K_{1,2} & \cdots & K_{1,n-2} & K_{1,n-1} & K_{1,n} \\ K_{2,0} & K_{2,1} & K_{2,2} & & K_{2,n-2} & K_{2,n-1} & K_{2,n} \\ & \vdots & & \ddots & & \vdots & \\ K_{n-2,0} & K_{n-2,1} & K_{n-2,2} & & K_{n-2,n-2} & K_{n-2,n-1} & K_{n-2,n} \\ K_{n-1,0} & K_{n-1,1} & K_{n-1,2} & \cdots & K_{n-1,n-2} & K_{n-1,n-1} & K_{n-1,n} \\ K_{n,0} & K_{n,1} & K_{n,2} & & K_{n,n-2} & K_{n,n-1} & K_{n,n} \end{bmatrix}$$

Assim, para um nó qualquer (*Ni*) com três deslocabilidades, tem-se as seguintes regras de correspondência:

Quadro 01 – Relações de correspondência.

	$\boldsymbol{N_i}$
x	$3.n_i$
y	$3.n_i + 1$
z	$3.n_i + 2$

2.12.9 Sistema de compatibilidade

O sistema de equações de compatibilidade para uma determinada estrutura, constituído pelos elementos apresentados ao longo desta seção, pode representado da seguinte maneira:

$$\begin{Bmatrix} R_0 \\ R_1 \\ R_2 \\ \vdots \\ R_{n-2} \\ R_{n-1} \\ R_n \end{Bmatrix} = \begin{Bmatrix} P_0 \\ P_1 \\ P_2 \\ \vdots \\ P_{n-2} \\ P_{n-1} \\ P_n \end{Bmatrix} + \begin{Bmatrix} F_0 \\ F_1 \\ F_2 \\ \vdots \\ F_{n-2} \\ F_{n-1} \\ F_n \end{Bmatrix} + \begin{bmatrix} K_{0,0} & K_{0,1} & K_{0,2} & \cdots & K_{0,n-2} & K_{0,n-1} & K_{0,n} \\ K_{1,0} & K_{1,1} & K_{1,2} & \cdots & K_{1,n-2} & K_{1,n-1} & K_{1,n} \\ K_{2,0} & K_{2,1} & K_{2,2} & \cdots & K_{2,n-2} & K_{2,n-1} & K_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ K_{n-2,0} & K_{n-2,1} & K_{n-2,2} & \cdots & K_{n-2,n-2} & K_{n-2,n-1} & K_{n-2,n} \\ K_{n-1,0} & K_{n-1,1} & K_{n-1,2} & \cdots & K_{n-1,n-2} & K_{n-1,n-1} & K_{n-1,n} \\ K_{n,0} & K_{n,1} & K_{n,2} & \cdots & K_{n,n-2} & K_{n,n-1} & K_{n,n} \end{bmatrix} \begin{Bmatrix} D_0 \\ D_1 \\ D_2 \\ \vdots \\ D_{n-2} \\ D_{n-1} \\ D_n \end{Bmatrix}$$

2.12.10 Diagramas de esforços

Para determinar os diagramas da estrutura, em primeiro lugar, deve-se computar os esforços internos de cada barra nos seus respectivos sistemas de coordenadas locais. Para tanto, faz-se o procedimento inverso ao realizado para determinar os vetores e matrizes globais.

Assim, deve-se definir $\{f\}$, $\{d\}$ e $[k]$ para tornar possível a resolução das equações de compatibilidade locais e a determinação do valor das reações locais $\{r\}$ nos apoios de cada uma das barras. Segundo Kassimali (2012), as reações locais podem ser obtidas por meio da seguinte relação:

$$\{r\} = \{f\} + [k]\{d\}$$

em que o vetor $\{d\}$ é calculado por meio da projeção do vetor de deslocamento da barra no sistema global no sistema local, isto é:

$$\{d\} = [T_\theta]\{d_g\}.$$

De posse dos valores de $\{r\}$, descobre-se as reações dos esforços nas barras, que coincidem com o valor do somatório dos esforços internos atuando em determinado ponto.

3. METODOLOGIA

A abordagem utilizada no desenvolvimento da ferramenta computacional para análise de estruturas reticuladas planas pode ser decomposta em quatro etapas principais: revisão bibliográfica, estruturação do aplicativo, desenvolvimento da ferramenta e verificação dos resultados. Esta seção tem a finalidade de descrever a metodologia adotada ao longo da criação da ferramenta computacional, definindo o tipo de pesquisa realizado, os procedimentos efetuados para a coleta de requisitos e os mecanismos para obtenção e análise dos resultados gerados pelo sistema.

3.1 Revisão bibliográfica

A pesquisa foi elaborada a partir de uma revisão bibliográfica dos principais autores e pesquisadores do assunto, com o objetivo de identificar as principais dificuldades encontradas no desenvolvimento de sistemas voltados à análise de estruturas reticuladas, bem como de organizar um referencial teórico que pudesse auxiliar na criação da ferramenta computacional, no caso de eventuais dúvidas. Para tanto, utilizou-se como fontes de pesquisa: livros, artigos e teses de dissertação.

3.2 Estruturação do aplicativo

Durante a revisão bibliográfica, buscou-se reunir o máximo possível de informações sobre o método dos elementos finitos e todas as ferramentas que pudessem auxiliar no desenvolvimento do aplicativo. De posse dessas informações, foi possível definir quais requisitos o aplicativo deveria atender; qual modelo matemático deveria ser adotado para a representação e análise das estruturas; e como deveria ser a sua estrutura gráfica. Além disso, também foram analisadas as possíveis limitações da ferramenta, para que o método de cálculo pudesse ser adaptado a estas.

3.2.1 Requisitos da ferramenta

Esta etapa teve como objetivo a definição dos limites do sistema, bem como de suas principais funcionalidades, que fornecem uma base sólida para que se possa estimar os esforços de desenvolvimento da ferramenta.

3.2.1.1 Requisitos não-funcionais:

a) desempenho: a ferramenta deve ser capaz de calcular os esforços e deformações da estrutura em até 1000ms;

b) usabilidade: a ferramenta deve ser capaz de exibir graficamente a estrutura deformada; possibilitando a ampliação (zoom) de determinado ponto da estrutura;

c) confiabilidade: a ferramenta não deve apresentar erros de cálculo; e

d) compatibilidade: a ferramenta deve ser compatível com a versão 3.0 da linguagem de programação Python.

3.2.1.2 Requisitos funcionais:

a) relativos à entrada de dados:

I. a ferramenta deve permitir a definição das localizações dos nós da estrutura, especificadas por meio de coordenadas geométricas em um plano bidimensional.

II. a ferramenta deve permitir a definição das propriedades elásticas dos materiais;

III. a ferramenta deve permitir a definição das propriedades das seções de cada

barra da estrutura;

IV. a ferramenta deve permitir a definição da geometria da estrutura, bem como a configuração das barras que a compõem.

V. a ferramenta deve permitir a definição das condições de restrição dos apoios; e

VI. a ferramenta deve permitir a definição das cargas que atuam na estrutura, tanto concentradas quanto cargas distribuídas.

b) relativos ao processamento de dados:

I. a ferramenta deve ser capaz de calcular o número de nós informados pelo usuário;

II. a ferramenta deve ser capar de identificar nós coincidentes, isto é, que possuem as mesmas coordenadas geométricas;

III. a ferramenta deve ser capaz de calcular o número de barras informadas pelo usuário;

IV. a ferramenta deve ser capaz de identificar barras coincidentes, isto é, aquelas que possuem exatamente os mesmos nós de início e fim;

V. a ferramenta deve ser capaz de calcular o grau de liberdade da estrutura;

VI. a ferramenta deve ser capaz de lidar com multiplicação de matrizes e resolução de sistemas lineares; e

VII. a ferramenta deve ser capaz de realizar o processamento em separado de cada elemento da estrutura, por meio de suas respectivas matrizes de rigidez.

c) relativos à saída dos resultados:

I. a ferramenta deve ser capaz de exportar os resultados do processamento, isto é, esforços internos, deslocamentos, matrizes de rigidez, etc.;

II. a ferramenta deve permitir a visualização dos deslocamentos nodais da estrutura informada pelo usuário;

III. a ferramenta deve permitir a visualização das reações de apoio da estrutura informada pelo usuário;

IV. a ferramenta deve permitir a visualização do diagrama de momentos fletores da estrutura informada pelo usuário;

V. a ferramenta deve permitir a visualização do diagrama de esforços cortantes da estrutura informada pelo usuário; e

VI. a ferramenta deve permitir a visualização do diagrama de esforços normais da estrutura informada pelo usuário.

3.3 Programação do aplicativo

Definida toda a parte teórica do trabalho, deu-se início à parte prática, que resultou na codificação da ferramenta. Esta etapa buscou atender às necessidades técnicas, gráficas e matemáticas definidas no tópico anterior, da forma mais eficiente o possível. O paradigma de programação adotado foi o orientado a objetos, pois a codificação fica mais próxima do cenário real do problema a ser resolvido e facilita a reutilização de componentes.

3.4 Análise dos resultados

A última etapa consistiu na validação dos resultados apresentados pelo sistema. Esta fase teve a finalidade de encontrar, documentar e endereçar os defeitos existentes na ferramenta desenvolvida, comparando-se os resultados obtidos por meio dela com os dados advindos de solução disponível no meio acadêmico (Ftool).

4. ARQUITETURA DO SISTEMA

A primeira etapa após a definição dos requisitos funcionais e não-funcionais da aplicação consistiu na definição da arquitetura do sistema, isto é, os componentes que formam o sistema e os mecanismos de interação existentes entre estes componentes. A arquitetura foi concebida com o objetivo de reduzir a complexidade da aplicação, por meio de abstração e separação de interesses.

A arquitetura pode ser definida como o núcleo da ferramenta, pois serve de base para a concepção das características mais importantes do sistema, interligando bibliotecas matemáticas capazes de processar operações matriciais, resolver sistemas lineares, realizar a interpolação paramétrica de pontos, desenhar diagramas e apresentar interfaces gráficas aos usuários.

Neste capítulo serão descritos os principais componentes da ferramenta de análise estrutural, isto é, aqueles que serão utilizados para representar os elementos da estrutura a ser analisada, tais como nós, barras, materiais, seções, apoios e cargas; e os objetos que irão compor a parte visual da solução formando a sua interface gráfica, que possibilitará a interação dos usuários com a aplicação.

4.1 Elementos constituintes da estrutura

Os componentes descritos nesta seção foram definidos com base nos conceitos apresentados no item 2.1 deste trabalho. Dessa forma, foram criadas classes para representar os seguintes elementos: material, seção, nó, barra, carga concentrada, carga distribuída, apoio, etc. Estas classes se relacionam entre si, o que possibilita a troca de informações e uma melhor definição de responsabilidades.

Em orientação a objetos, o paradigma de programação adotado neste trabalho, uma classe é uma estrutura que abstrai um conjunto de objetos com características similares. As classes definem o comportamento de seus objetos por meio de métodos (funções) e as possíveis características destes objetos por meio de atributos. Estas estruturas de dados possuem um comportamento padrão, denominado de construtor, que é invocado no momento da criação de um objeto de uma classe.

4.1.1 Classe Material

Por conta das simplificações adotadas neste trabalho, a Classe Material pode ser representada por meio de poucas variáveis. A premissa de que serão utilizados exclusivamente materiais isotrópicos e homogêneos simplifica o cálculo da estrutura, tornando necessárias apenas as informações do módulo de elasticidade E e da massa específica ρ do material.

Neste momento, a massa específica ρ não terá influência na análise das estruturas, porém, futuramente, poderá ser utilizada para o cômputo de cargas de peso próprio calculadas a partir da área da seção transversal das barras.

Os parâmetros que compõem o construtor desta classe são: o módulo de elasticidade (E) e a massa específica (ρ). Quanto aos seus métodos, importa mencionar apenas os seguintes: *__eq__*, que verifica se dois materiais são iguais, isto é, se possuem as mesmas características; e *__str__*, que cria uma representação inteligível das características do material.

4.1.2 Classe Seção

A Classe Seção representa o formato da seção transversal de barras da estrutura, possuindo as informações de área e inércia como suas características principais. Neste trabalho, adota-se a hipótese de utilização de seções genéricas, para as quais o usuário deverá informar

tanto a área quanto a inércia. Sendo assim, estes valores não serão computados pelo sistema, mas sim utilizados como parâmetros de entrada.

Os parâmetros que compõem o construtor desta classe são: a área da seção (*A)* e a inércia (*I)*. Quanto aos seus métodos, importa mencionar apenas os seguintes: *__eq__*, que verifica se duas seções genéricas são iguais, isto é, se possuem as mesmas características; e *__str__*, que cria uma representação inteligível das características da seção.

4.1.3 Classe Apoio

A Classe Apoio corresponde às restrições de deslocamento aplicadas aos nós. Estas restrições podem impedir deslocamentos (eixos *x* e *y*) ou rotações (eixo *z*). Dessa forma, este elemento pode ser representado por meio de três variáveis distintas, que determinam se as deslocabilidades estão livres ou restringidas nos eixos *x*, *y* e *z*. Portanto, existem seis possíveis combinações para descrever os tipos de apoio.

Os parâmetros que compõem o construtor desta classe são: o deslocamento em *x* (fixo ou livre); o deslocamento em *y* (fixo ou livre); e a rotação em *z* (fixa ou livre). Quanto aos seus métodos, importa mencionar apenas o seguinte: *__str__*, que cria uma representação inteligível das características do apoio.

4.1.4 Classe Nó

A Classe Nó é representada por três características principais: as coordenadas cartesianas *x* e *y*; as restrições de apoio nele configuradas; e a carga pontual que nele atua. Além disso, este elemento possui um identificador, que servirá para determinar os extremos das barras.

Os parâmetros que compõem o construtor desta classe são: as coordenadas *x* e *y*. Quanto aos seus métodos, importa mencionar apenas os seguintes: *__eq__*, que verifica se dois nós possuem as mesmas coordenadas; e *__str__*, que cria uma representação inteligível das características do nó.

4.1.5 Classe Barra

A Classe Barra é representada por um conjunto de características principais, tais como os índices de dois nós, um material e uma seção. Além destas, tem-se a definição dos

carregamentos distribuídos e a configuração das rótulas da barra. Assim como no caso dos Nós, este elemento possui um identificador que permite determinar cada uma das Barras.

Os parâmetros que compõem o construtor desta classe são: os índices dos nós, do material e da seção. Quanto aos seus métodos, importa mencionar apenas os seguintes: *__eq__*, que verifica se duas barras são idênticas; *__str__*, que cria uma representação inteligível das características do nó; *gerar_matriz_rigidez_local*, que cria a matriz de rigidez da barra de acordo com a configuração das rótulas; e *gerar_matriz_transformacao*, que cria a matriz de transformação da barra de acordo com as coordenadas de seus nós.

4.1.6 Classe Carga Pontual

A Classe Carga Pontual representa os carregamentos localizados em um determinado ponto da estrutura, que podem agir em três direções das estruturas planas: uma força no eixo *x*, uma força no eixo *y* e um momento fletor no eixo *z*. Neste trabalho, cada nó possuirá uma única carga pontual, assim o objeto deverá permitir a inserção simultânea de carregamentos nestas três direções.

Os parâmetros que compõem o construtor desta classe são: o índice do nó, a força P_x, a força P_y e o momento M_z. Quanto aos seus métodos, importa mencionar apenas o seguinte: *__str__*, que cria uma representação inteligível da carga pontual.

4.1.7 Classe Carga Distribuída

A Classe Carga Distribuída representa os carregamentos distribuídos ao longo de uma determinada barra. Neste trabalho, as cargas distribuídas atuam nos eixos locais *x* e *y* das barras, podendo apresentar valores de carregamento diferentes no início e no final da barra.

Além disso, considera-se que as cargas distribuídas se distribuem linearmente ao longo da barra, sendo suficiente para determinar a curva de distribuição apenas os valores inicial e final dos carregamentos. Assim, para definir a carga distribuída, faz-se necessário estabelecer os valores dos carregamentos Q_{x1} e Q_{y1}, localizados sobre o primeiro nó, nos eixos x e y; e os valores dos carregamentos Q_{x2} e Q_{y2}, localizados sobre o segundo nó, nos eixos x e y.

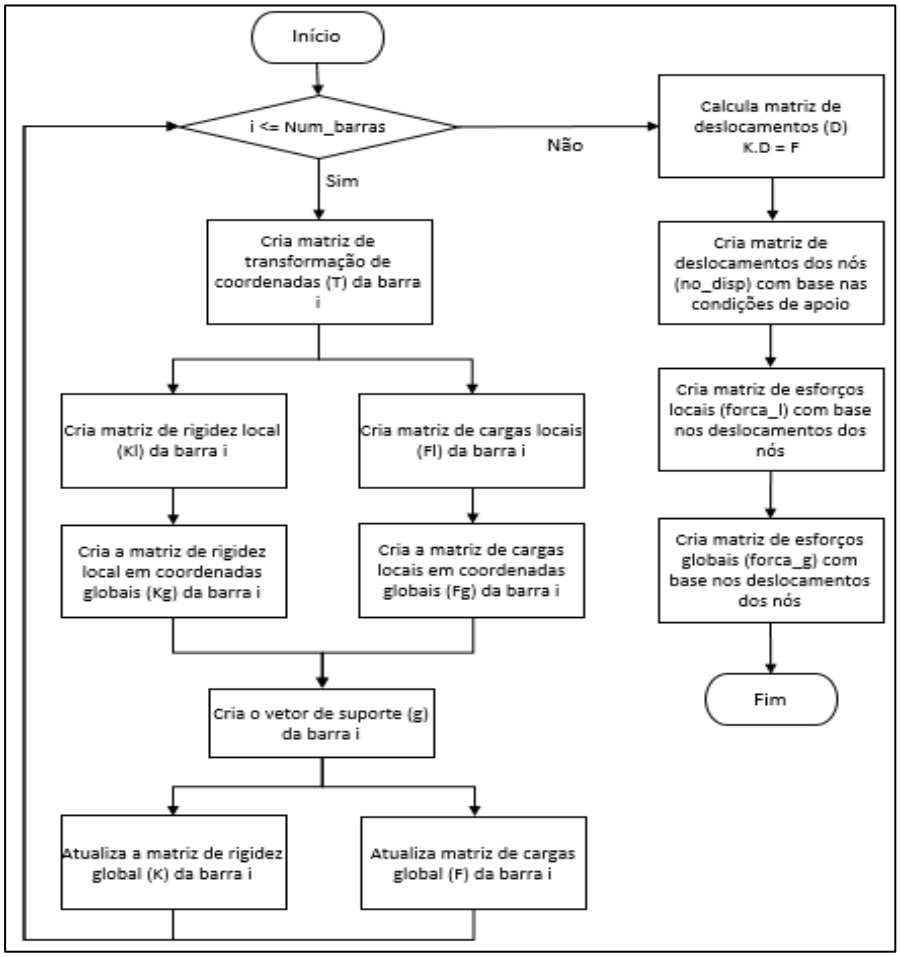
Os parâmetros que compõem o construtor desta classe são: o índice da barra e as intensidades do carregamento (Q_{x1}, Q_{x2}, Q_{y1}, Q_{y2}). Quanto aos seus métodos, importa mencionar apenas o seguinte: *__str__*, que cria uma representação inteligível da carga distribuída.

4.1.8 Classe Estrutura

A Classe Estrutura representa a estrutura reticulada propriamente dita, sendo formada por nós, barras, apoios e articulações. Assim, esta classe armazena os dados de nós, barras, materiais e seções informados pelos usuários do sistema. Além disso, este código é responsável por discretizar a estrutura, analisar os seus resultados e exportá-los.

Esta classe possui construtor padrão, isto é, sem parâmetros. Quanto aos seus métodos, importa mencionar apenas os seguintes: *__str__*, que cria uma representação inteligível da estrutura; *discretizar*, que cria uma estrutura subdividida, seguindo as condições e restrições da estrutura original, utilizada no cálculo das deformações das barras; *adicionar_no*, que adiciona um nó à estrutura; *adicionar_barra*, que adiciona uma barra à estrutura; *adicionar_material*, que adiciona um tipo de material à estrutura; *adicionar_secao*, que adiciona um tipo de seção à estrutura; e *calcular*, que segue o fluxograma descrito na Figura 16, elaborado com base nas rotinas propostas por Kassimali (2012).

Figura 16 – Fluxograma para análise de estruturas.



Fonte: Autor

4.2 Elementos constituintes da interface gráfica do usuário

Os componentes descritos nesta seção foram criados com o objetivo de proporcionar interatividade e facilidade de uso, possibilitar a entrada de dados e permitir a visualização dos resultados da análise de estruturas planas.

4.2.1 Classe Janela Principal (MainWindow)

A Classe Janela Principal é a responsável por criar a área visual que permite ao usuário interagir com a ferramenta de análise estrutural, bem como criar os elementos descritos na seção 2.1 deste trabalho, que compõem a estrutura. Esta tela agrega três componentes visuais: a barra de menu, a área de desenho e o menu lateral.

A barra de menu possibilita ao usuário interagir com as principais funcionalidades do sistema, isto é, salvar o desenho da estrutura, abrir desenhos já salvos anteriormente, exportar relatórios de análise, visualizar diagramas de deformação, diagramas de momentos fletores, diagramas de esforços cortantes e diagramas de esforços normais.

O menu lateral “Propriedades” está dividido em quatro abas: dados gerais, estrutura, cargas e restrições. A aba “Dados gerais” possibilita a entrada das características dos materiais e das seções. A aba “Estrutura” permite a entrada das coordenadas dos nós e das especificações das barras. A aba “Cargas” possibilita informar a intensidade das cargas concentradas e distribuídas que atuam na estrutura. A aba “Restrições” permite a configuração de articulações e apoios da estrutura.

A área de desenho, que será detalhada a seguir, é a responsável por exibir a estrutura informada pelo usuário, suas deformações e seus diagramas de esforços internos. Além disso, esta área permite ao usuário interagir com a estrutura, permitindo ao usuário exportar, ampliar, reduzir e centralizar a representação gráfica de uma determinada estrutura.

4.2.2 Classe Área de Desenho (Canvas)

A Classe Área de Desenho é responsável por criar um ambiente visual que permite ao usuário visualizar as informações previamente cadastradas da estrutura, podendo exibir a configuração a sua não deformada e os seus diagramas de deformações, de momentos fletores, de esforços cortantes e esforços normais. Todos os elementos visuais da estrutura (nós, barras, apoios, gráficos, etc.) são criados nesta classe por meio de figuras geométricas simples, tais

como pontos, retas e círculos. Além disso, esta área permite ao usuário interagir com a estrutura, possibilitando exportar, ampliar, reduzir, centralizar a imagem visualizada.

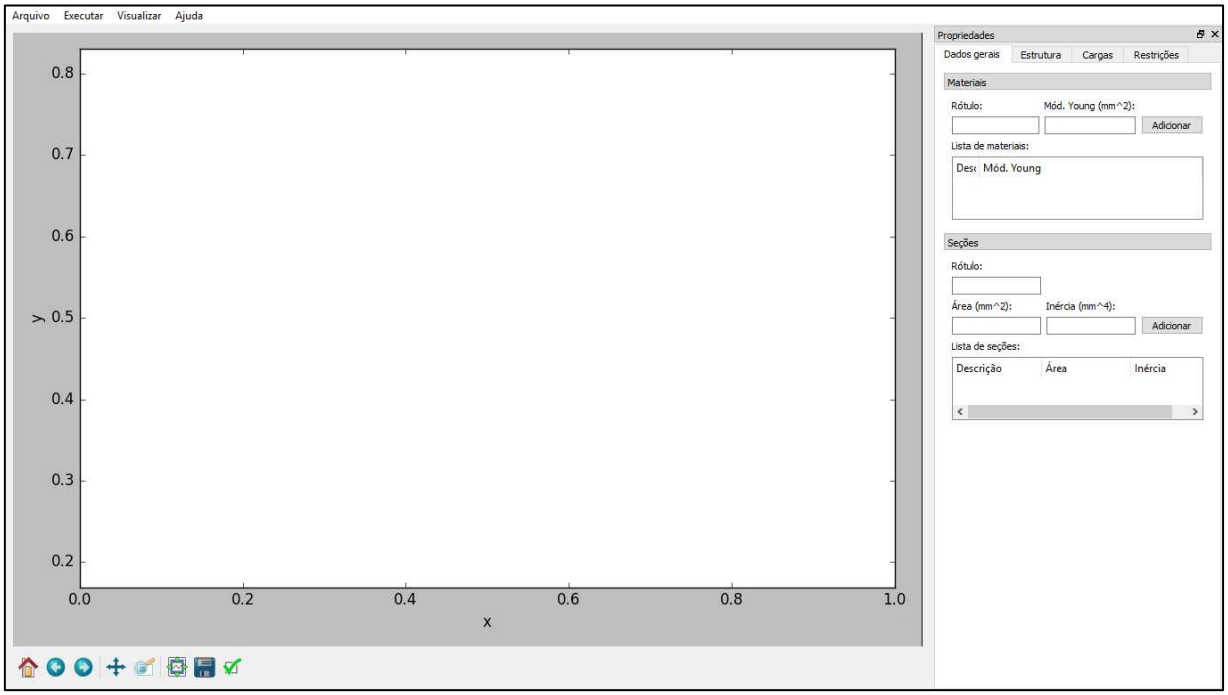
5. APRESENTAÇÃO DO SISTEMA

Este capítulo servirá como um guia rápido de utilização para a ferramenta desenvolvida, reunindo todas as funcionalidades do programa, bem como exemplificando o lançamento de um pórtico, desde a configuração de suas características até a visualização de seus resultados.

5.1 Visão geral

A tela inicial do sistema (Figura 17), exibida logo após o acionamento da ferramenta, apresenta algumas funcionalidades que possibilitam a inserção dos elementos estruturais, a abertura de arquivos criados anteriormente e a visualização dos resultados da análise.

Figura 17 – Visão geral da ferramenta.

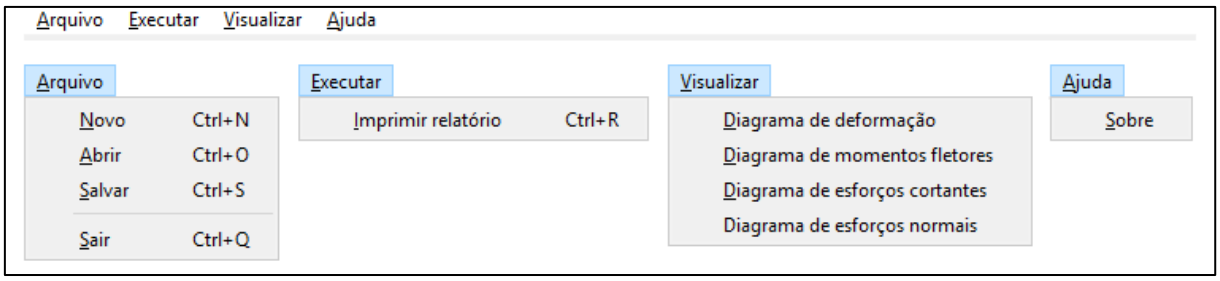


Fonte: Autor

Conforme explanado anteriormente, a tela inicial é composta por três itens principais: a barra de menu, o menu lateral de propriedades e a área de desenho.

A barra de menu (Figura 18) possibilita ao usuário interagir com as principais funcionalidades do sistema, isto é, salvar o desenho da estrutura, abrir desenhos já salvos anteriormente, exportar relatórios de análise, visualizar diagramas de deformação, diagramas de momentos fletores, diagramas de esforços cortantes e diagramas de esforços normais.

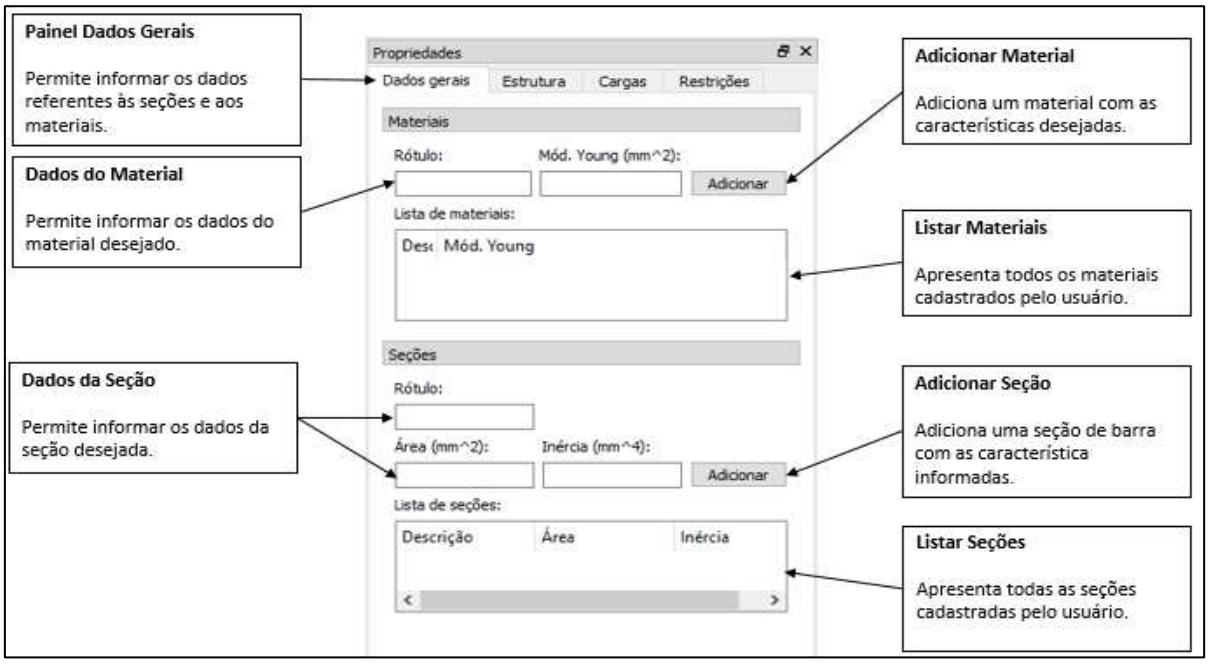
Figura 18 – Barra de menu da ferramenta.



Fonte: Autor

O painel “Propriedades” está dividido em quatro abas: Dados gerais, Estrutura, Cargas e Restrições. A aba “Dados gerais” (Figura 19) possibilita a especificação das características dos materiais e das seções transversais das barras.

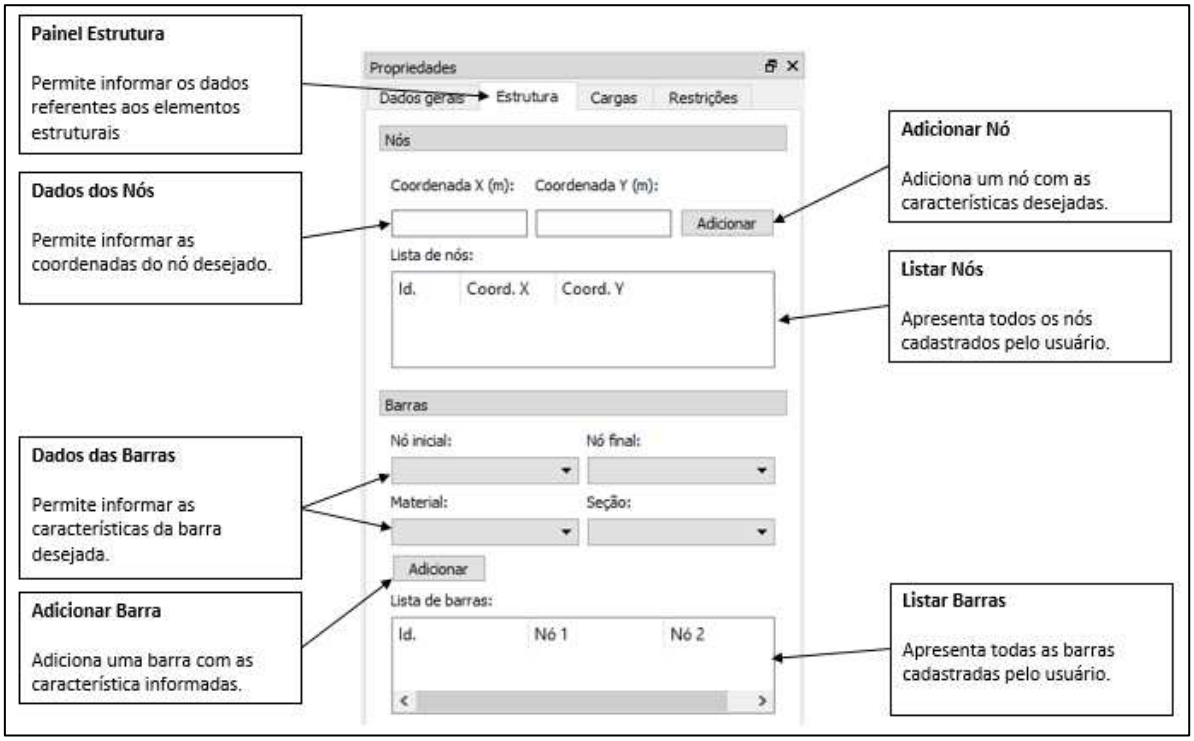
Figura 19 – Cadastro de materiais e seções da estrutura.



Fonte: Autor

A aba “Estrutura” (Figura 20) permite a entrada das coordenadas dos nós e das propriedades das barras, tais como suas extremidades, o tipo de material constituinte e o tipo de seção transversal.

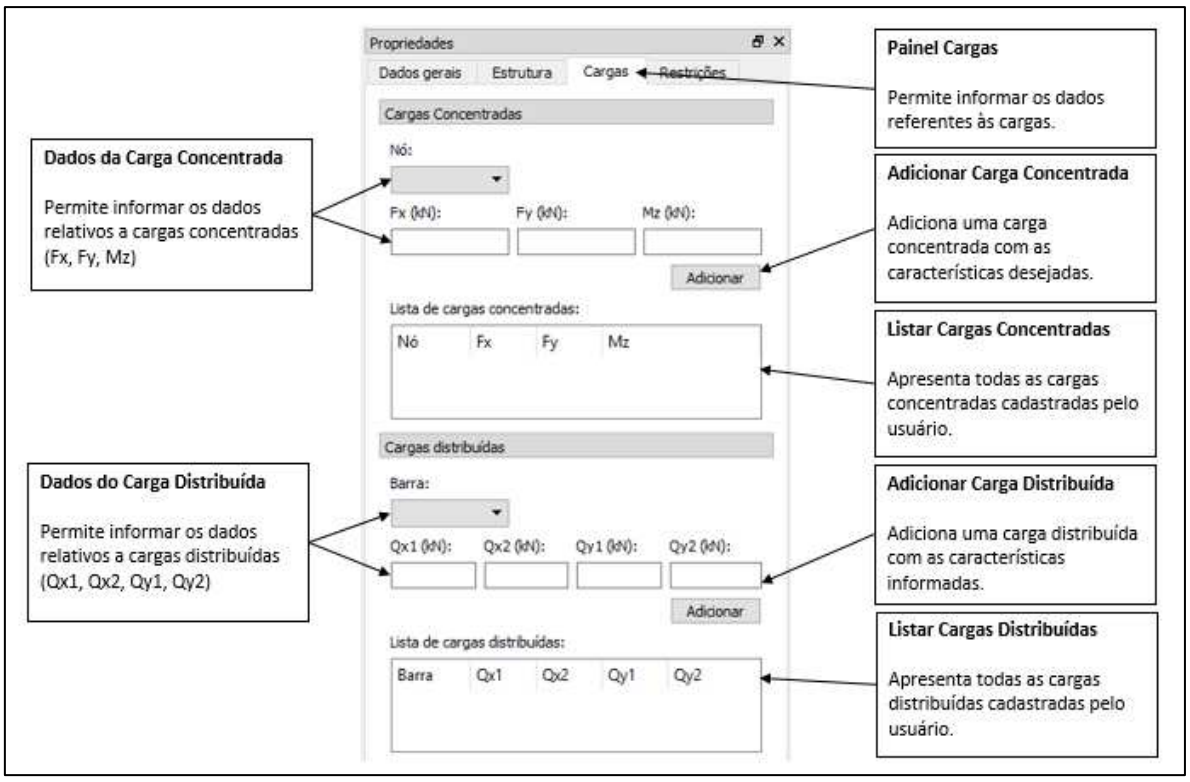
Figura 20 – Cadastro de nós e barras da estrutura.



Fonte: Autor

A aba “Cargas” (Figura 21) possibilita a inserção das intensidades das cargas concentradas e distribuídas que atuam na estrutura.

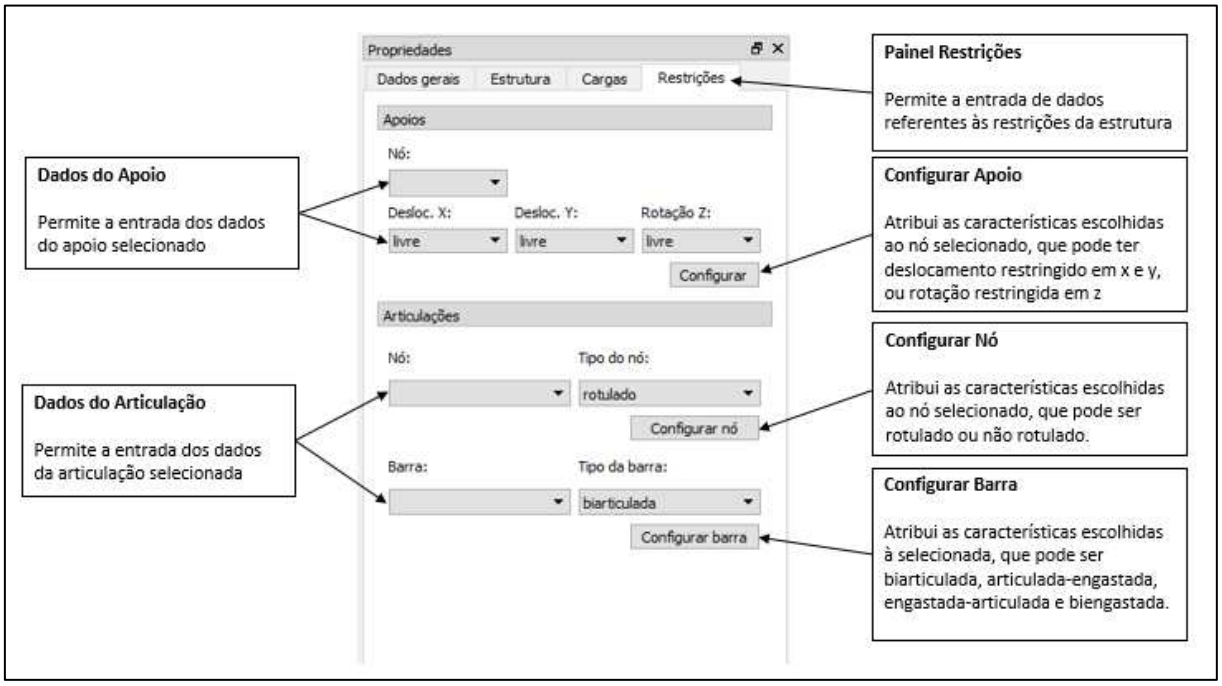
Figura 21 – Cadastro de cargas concentradas e distribuídas.



Fonte: Autor

A aba “Restrições” (Figura 22) permite a configuração dos apoios, aos quais os nós estão vinculados, e das articulações de uma estrutura, que podem estar associadas a nós ou extremidades de uma barra.

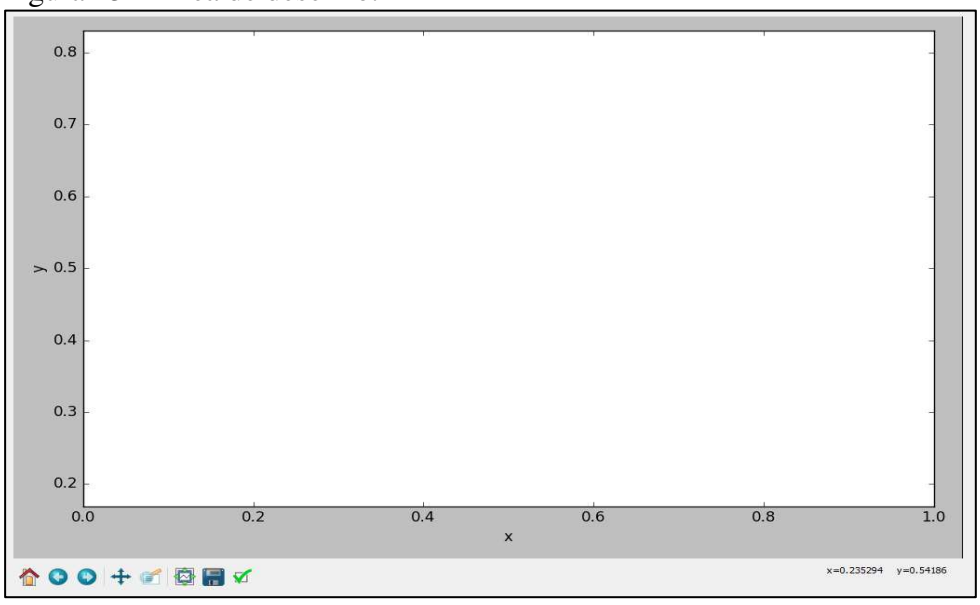
Figura 22 – Configuração das restrições.



Fonte: Autor

A Área de Desenho (Figura 23) exibirá a estrutura informada pelo usuário, suas deformações e seus diagramas de esforços internos. Além disso, esta área também permite a interação com a estrutura, por meio de comandos de ampliar, reduzir ou exportar a imagem.

Figura 23 – Área de desenho.



Fonte: Autor

5.2 Entrada de dados

O objetivo deste tópico é apresentar uma descrição pormenorizada de todos os passos necessários para a definição de uma estrutura similar à mostrada na Figura 24.

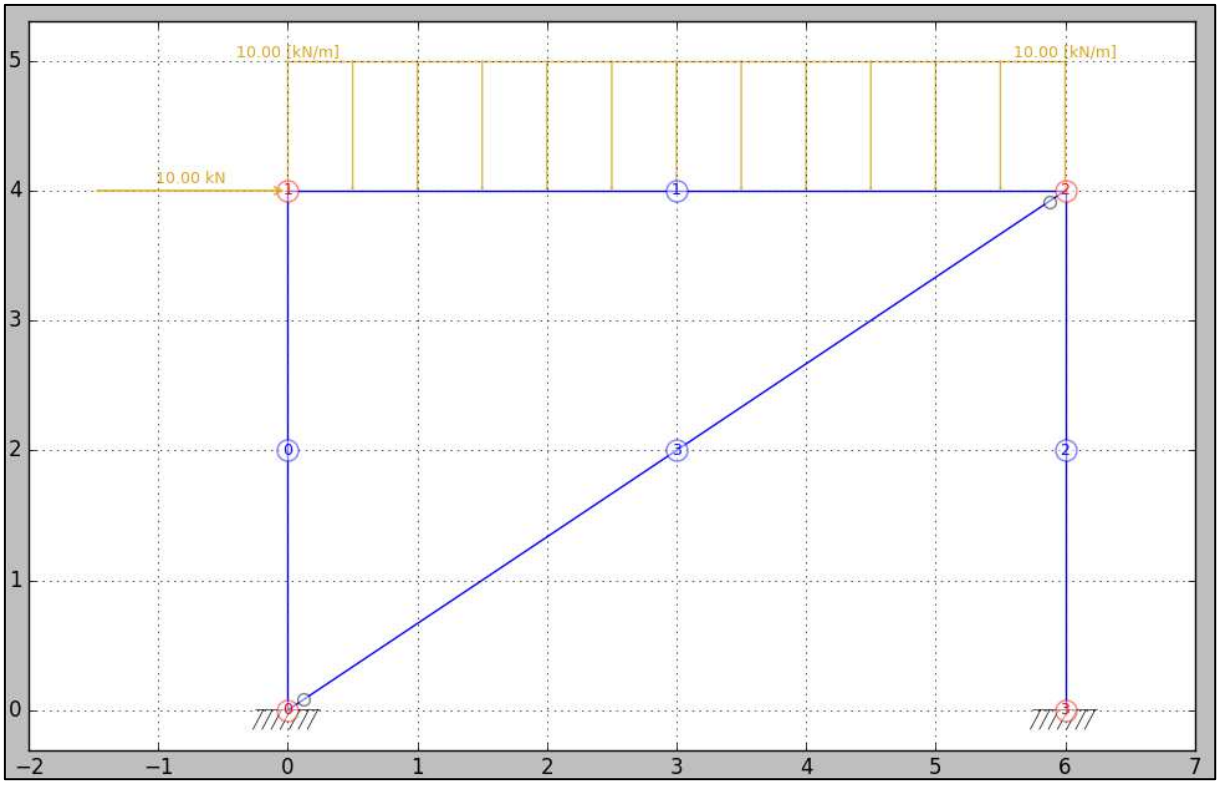
Para o presente exemplo, o material utilizado terá módulo de elasticidade (módulo de Young) equivalente a 200 GPa e a seção transversal terá área e momento de inércia iguais a 0.009 mm^2 e 0.0025 mm^4.

Os nós da estrutura estão localizados nos pontos (0,0); (0,4); (6,4) e (6,0), sendo indexados pelos números 0, 1, 2 e 3, respectivamente. Por sua vez, as barras conectam os nós de índices (0,1); (1,2); (2,3) e (0,2), sendo indexadas pelos valores 0, 1, 2, 3, respectivamente.

Deve-se acrescentar que as barras possuem especificações de material e seção uniformes, correspondentes às características dos elementos adicionados anteriormente. Além disso, quanto às restrições da estrutura, é importante esclarecer que a barra de índice 3 possui suas extremidades articuladas e que os nós de índices 0 e 3 encontram-se indeslocáveis.

Por fim, há uma carga concentrada horizontal de 10 kN aplicada ao nó de índice 1, e uma carga distribuída uniforme de 10 kN/m aplicada à barra de índice 1.

Figura 24 – Pórtico atirantado exemplificativo.



Fonte: Autor

5.2.1 Configuração de Materiais e Seções

Assim, na aba “Dados Gerais” do menu lateral “Propriedades” o usuário deverá informar uma descrição do material e o seu módulo de elasticidade, clicando em seguida no botão adicionar. Feito isso, o usuário deverá informar a área e a inércia da seção transversal e clicar no botão adicionar.

5.2.2 Inserção de nós

Após cadastrar os materiais e as seções transversais, o usuário deverá especificar os nós que compõem a geometria da estrutura. Isto pode ser feito por meio da aba “Estrutura”, do menu lateral “Propriedades”. Para tanto, o usuário deverá informar duas coordenadas de um plano cartesiano (x e y) e clicar no botão adicionar, fazendo com que o nó seja inserido no painel de desenho. Este processo deverá ser repetido quatro vezes, um para cada nó que faz parte da geometria da estrutura.

5.2.3 Adição de barras

Com os materiais, as seções transversais e os nós cadastrados, o usuário deverá informar as barras que compõem a geometria da estrutura. As barras também serão adicionadas por meio da aba “Estrutura” do menu lateral “Propriedades”. Para tanto, o usuário deverá selecionar dois nós (um de início e um de término), um tipo de material e um tipo de seção, todos já previamente cadastrados. Depois de entrar com esses dados, o usuário deverá clicar no botão adicionar para que a barra seja inserida no painel de desenho.

5.2.4 Configuração dos apoios

Após especificar as barras, o usuário deverá definir a configuração dos apoios. Isto poderá ser feito por meio da aba “Restrições”, do menu lateral “Propriedades”. Assim, o usuário deverá informar que os nós de índices 0 e 3 possuem tanto os deslocamentos em x e y quanto a rotação em z restringidos (fixos). Feito isso, deve-se clicar em configurar.

5.2.5 Configuração das rótulas

Tendo definido os apoios, o usuário deverá fazer a configuração das rótulas. Isto também será feito por meio da aba “Restrições”, do menu lateral “Propriedades”. Assim, o usuário deverá informar que a barra de índice 3 é biarticulada. Feito isso, clica-se em configurar barra.

5.2.6 Configuração das cargas

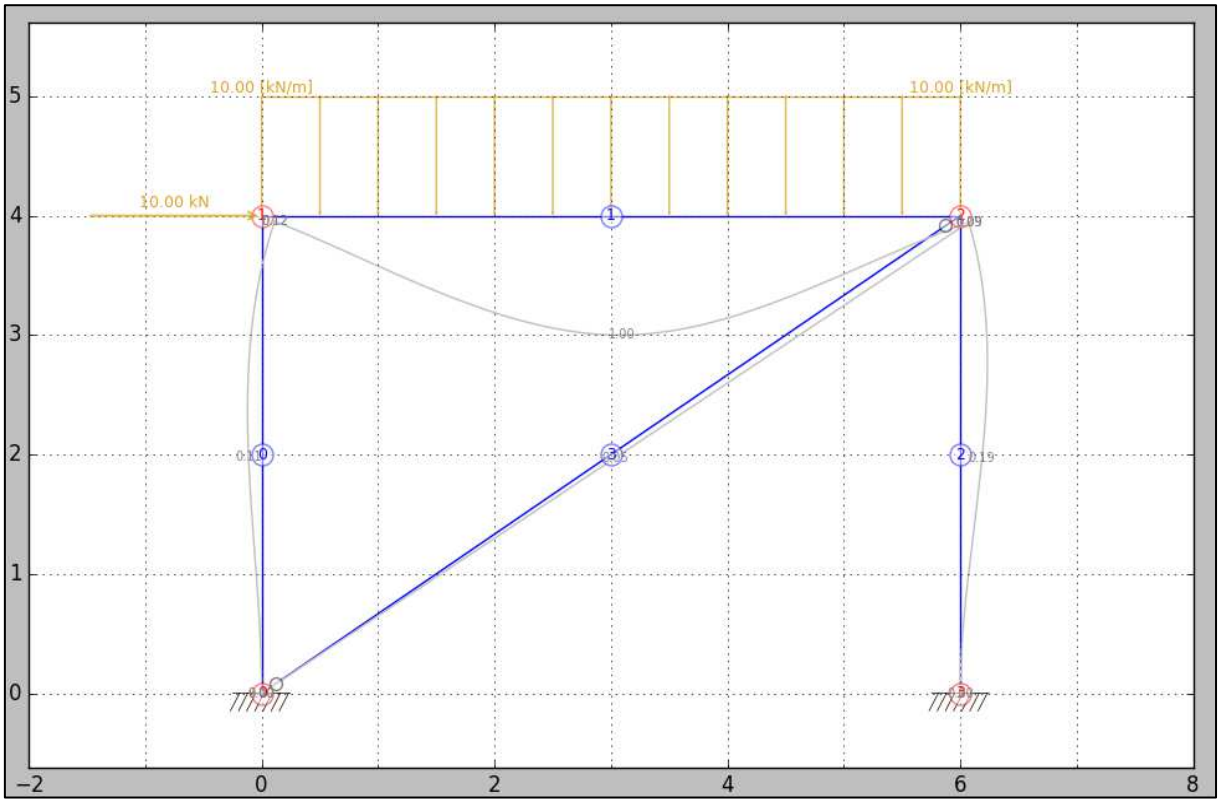
Definida a geometria da estrutura e suas restrições, o usuário deverá fazer a configuração das cargas. Isto será feito por meio da aba “Cargas”, do menu lateral “Propriedades”. Para o exemplo em análise, o usuário deverá atribuir uma carga concentrada de 10 kN no eixo *x* do nó de índice 1; e uma carga distribuída uniforme de 10 kN/m no eixo *y* da barra de índice 1.

5.3 Apresentação dos resultados

Com a estrutura adicionada, os resultados poderão ser acessados por meio de gráficos ou de relatórios. A parte gráfica permitirá que o usuário visualize os diagramas de deformação (Figura 25), momentos fletores (Figura 26), esforços normais (Figura 27) e esforços cortantes (Figura 28), bem como os valores das reações da estrutura (Figura 29). Estas funcionalidades gráficas podem ser acessadas por meio do menu “Visualizar” da barra de menu.

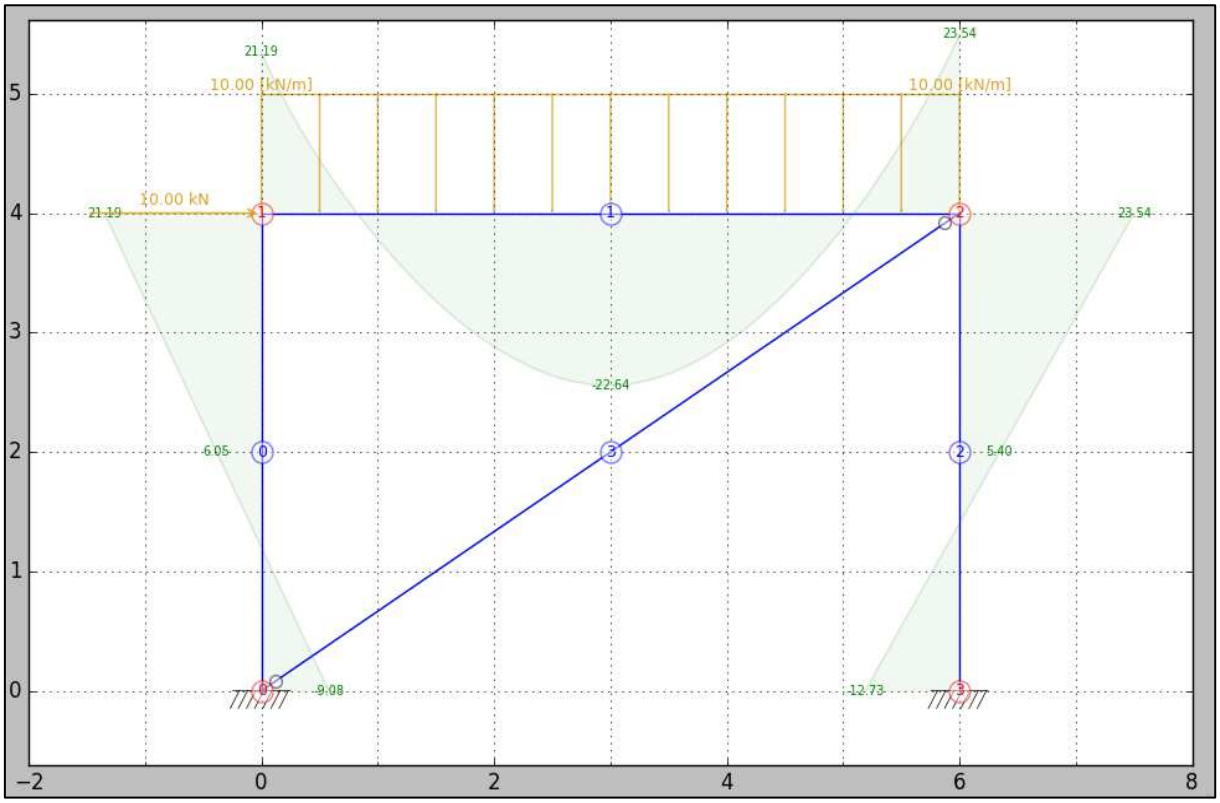
Por sua vez, o relatório de análise (Figura 30) pode ser acessado por meio do menu “Executar - Imprimir relatório”. Este relatório apresenta o vetor global de forças, o vetor solução de deslocamentos nodais, os deslocamentos nodais propriamente ditos e as cargas nas extremidades das barras, tanto em coordenadas locais quanto em coordenadas globais.

Figura 25 – Diagrama de deformação.



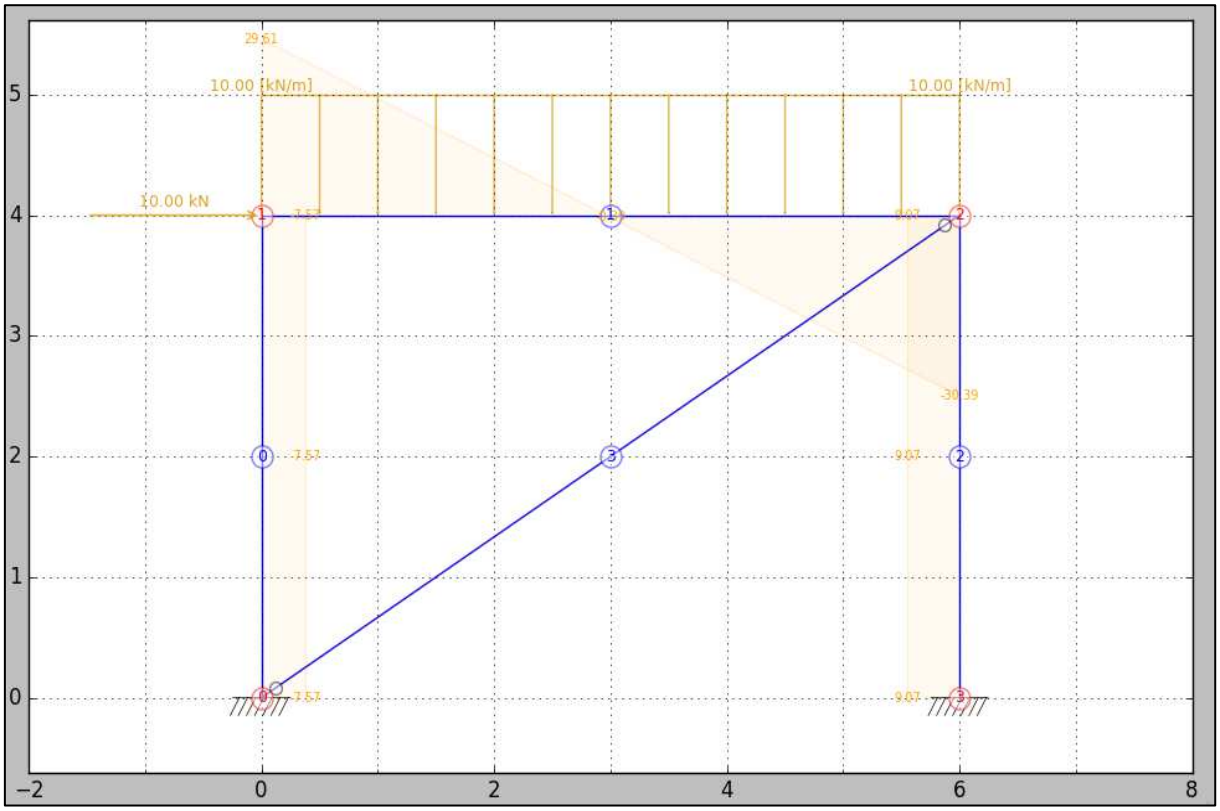
Fonte: Autor

Figura 26 – Diagrama de momentos fletores.



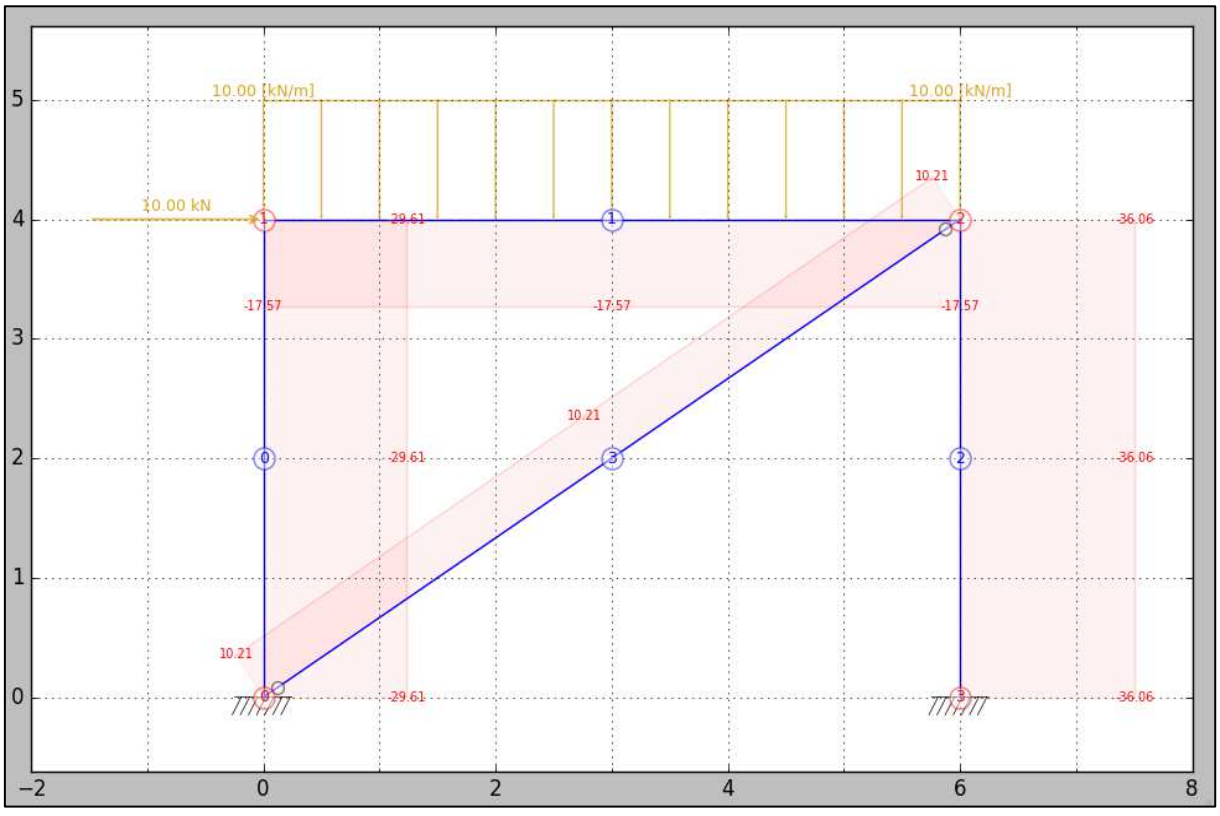
Fonte: Autor

Figura 27 – Diagrama de esforços cortantes.



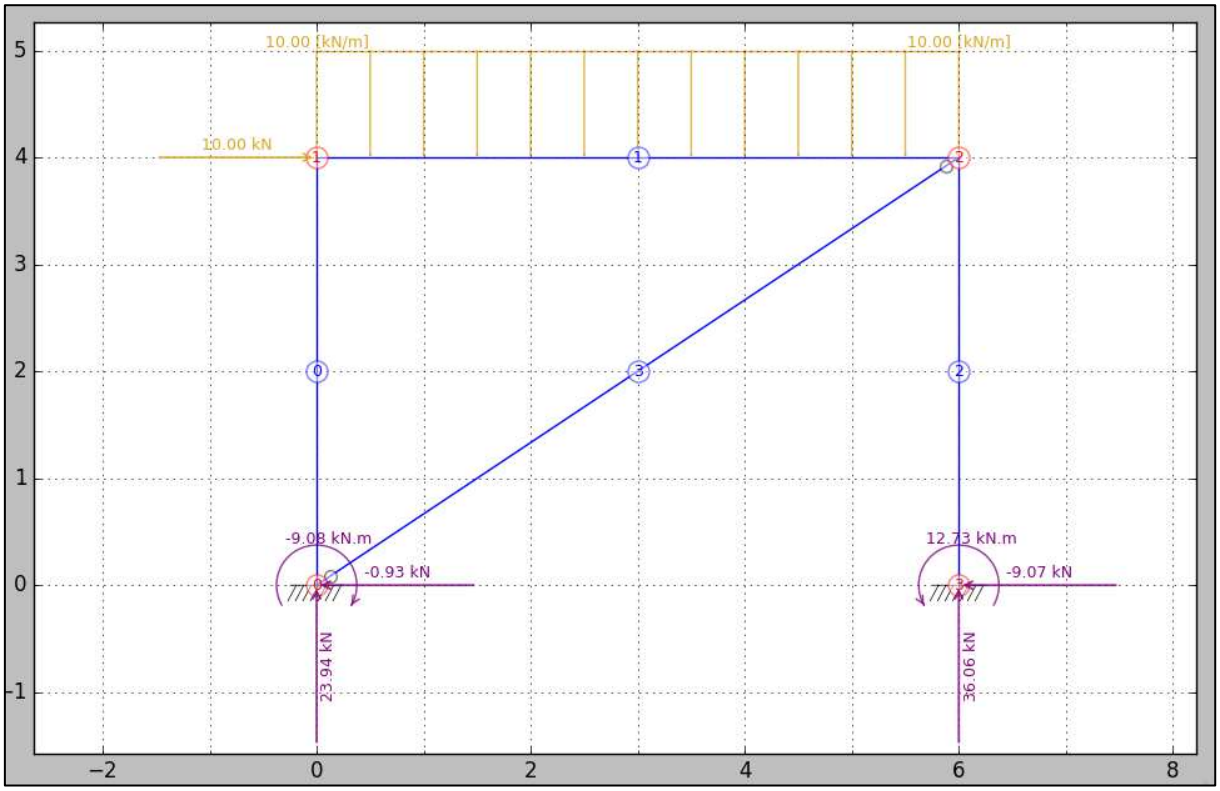
Fonte: Autor

Figura 28 – Diagrama de esforços normais.



Fonte: Autor

Figura 29 – Valores das reações nos apoios da estrutura.



Fonte: Autor

Figura 30 – Relatório de análise do pórtico exemplificativo.

```
Imprimindo os resultados da análise estrutural:

1. Vetor global de forças (F):

+-----------+
|     F     |
+-----------+
|      10.00|
|     -30.00|
|     -30.00|
|       0.00|
|     -30.00|
|      30.00|
+-----------+

2. Vetor solução de deslocamentos nodais (Delta):

+---------------+
|     Delta     |
+---------------+
|     0.00016116|
|    -0.00006580|
|    -0.00048416|
|     0.00010260|
|    -0.00008013|
|     0.00043231|
+---------------+

3. Deslocamentos nodais:

+----+----------------+----------------+----------------+
| No | Deslocamento X | Deslocamento Y |    Rotação Z   |
+----+----------------+----------------+----------------+
|   1|      0.00000000|      0.00000000|      0.00000000|
|   2|      0.00016116|     -0.00006580|     -0.00048416|
|   3|      0.00010260|     -0.00008013|      0.00043231|
|   4|      0.00000000|      0.00000000|      0.00000000|
+----+----------------+----------------+----------------+
```

4. Ações nas barras em coordenadas locais:

Barra	Fx1	Fy1	Mz1	Fx2	Fy2	Mz2
1	29.6078	-7.5671	-9.0822	-29.6078	7.5671	-21.1861
2	17.5671	29.6078	21.1861	-17.5671	30.3922	-23.5394
3	36.0584	9.0678	23.5394	-36.0584	-9.0678	12.7316
4	-10.2149	0.0000	0.0000	10.2149	0.0000	0.0000

5. Ações nas barras em coordenadas globais:

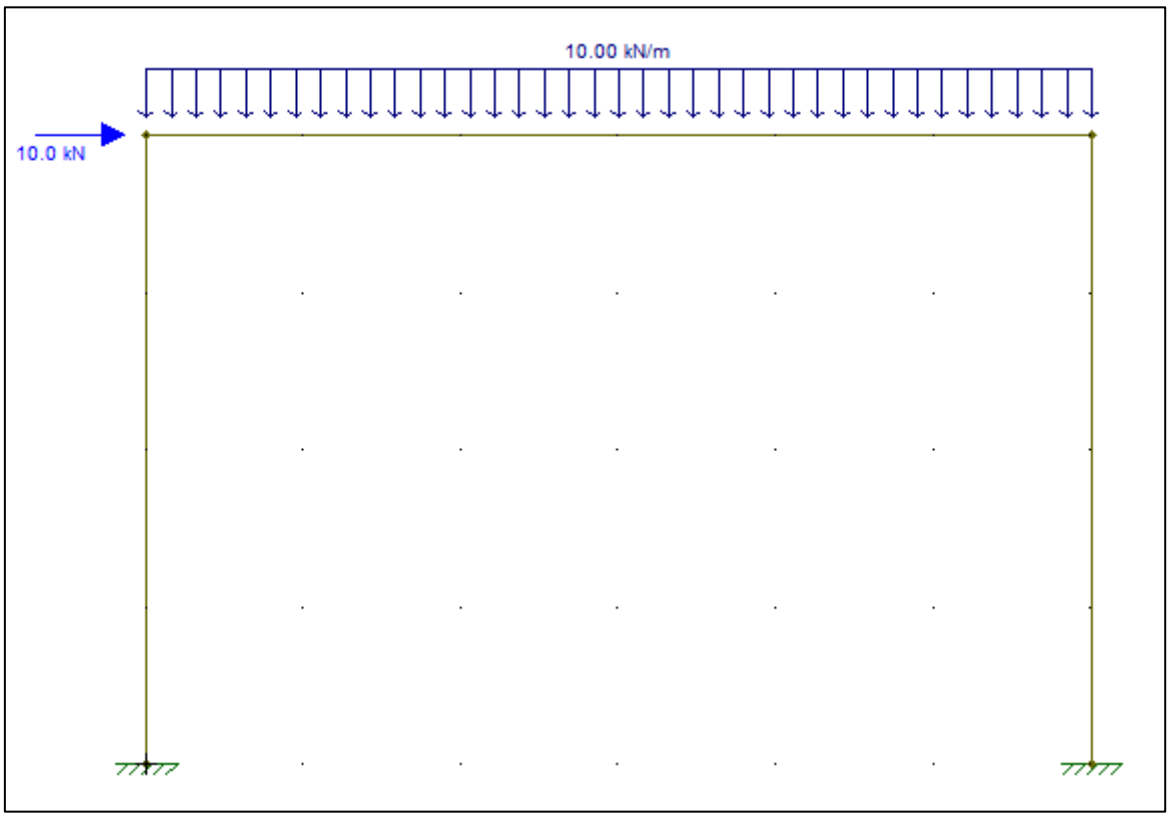
Barra	Fx1	Fy1	Mz1	Fx2	Fy2	Mz2
1	7.5671	29.6078	-9.0822	-7.5671	-29.6078	-21.1861
2	17.5671	29.6078	21.1861	-17.5671	30.3922	-23.5394
3	9.0678	-36.0584	23.5394	-9.0678	36.0584	12.7316
4	-8.4993	-5.6662	0.0000	8.4993	5.6662	0.0000

Fonte: Autor

5.4 Validação dos resultados

Para validação dos resultados, o seguinte pórtico (Figura 31) foi analisado tanto na ferramenta desenvolvida neste trabalho quanto no sistema educacional denominado de Ftool 3.01. De posse dos dados numéricos apresentados pelas duas aplicações, fez-se a comparação de seus resultados por inspeção visual, que comprovou a correção do sistema desenvolvido.

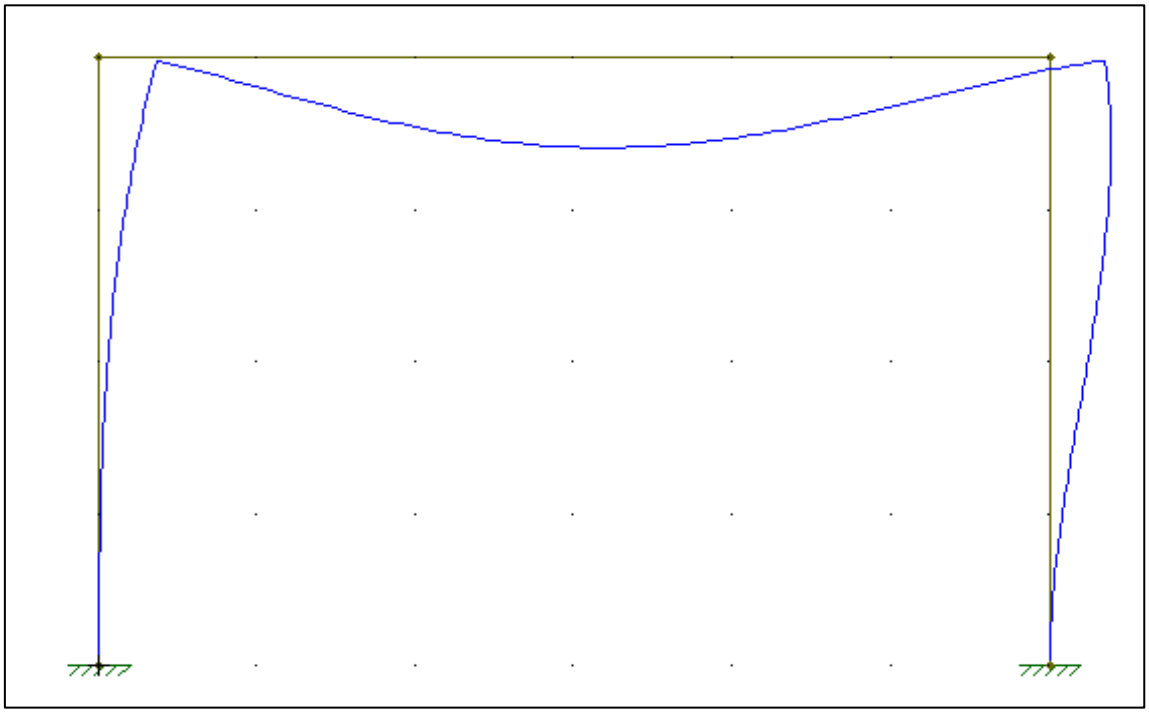
Figura 31 – Pórtico a ser validado.



Fonte: Autor

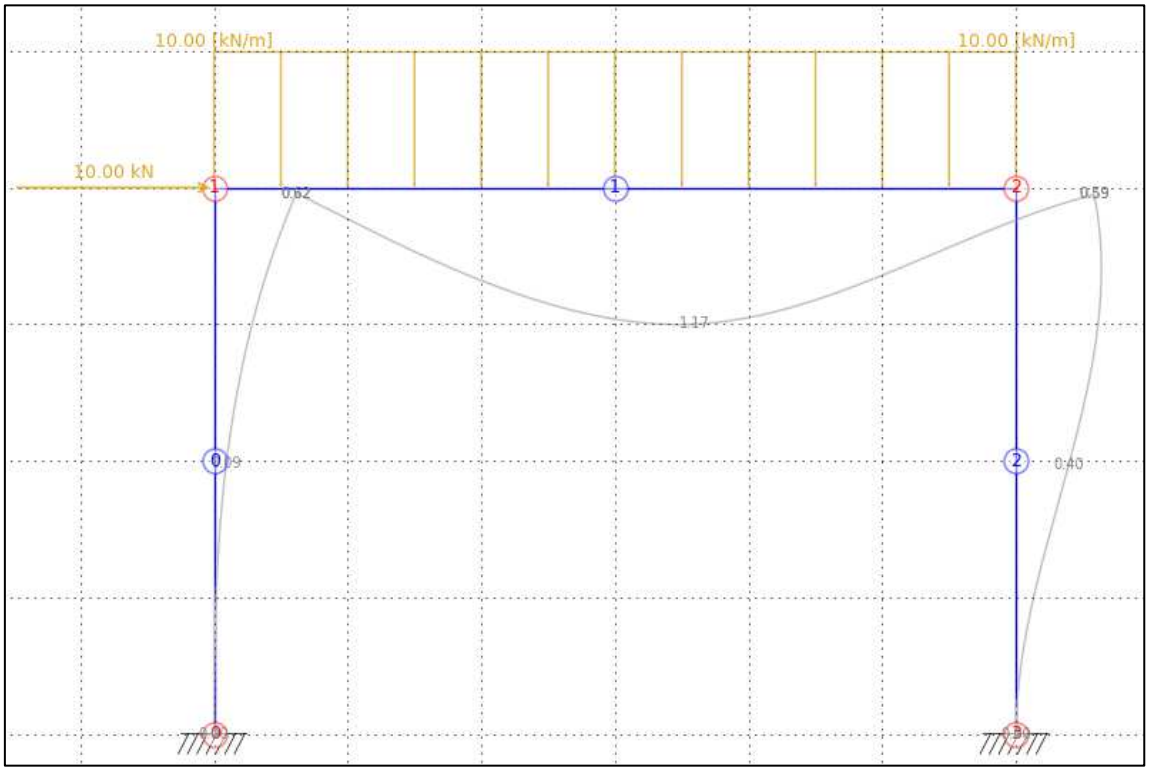
Nas Figuras 32 e 33, são apresentados os diagramas de deformação da estrutura, gerados pelo Ftool 3.01 e pela ferramenta desenvolvida neste trabalho, respectivamente.

Figura 32 – Configuração deformada da estrutura (Ftool).



Fonte: Autor

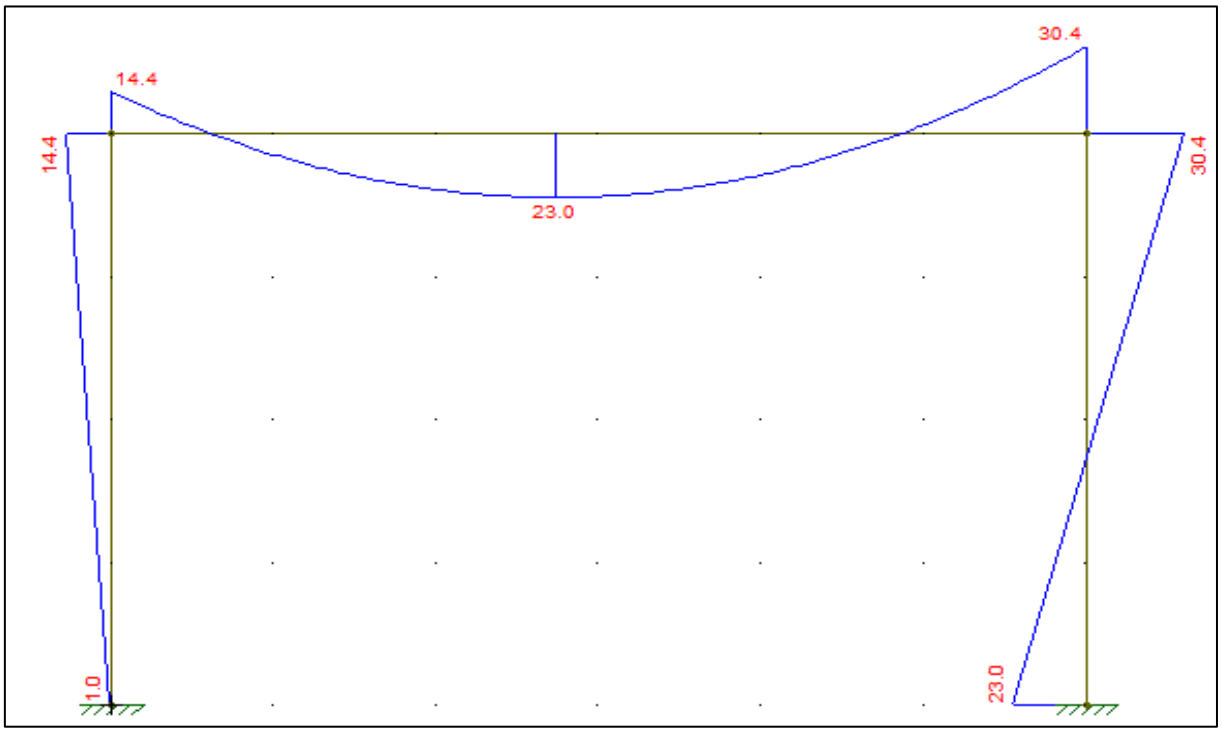
Figura 33 – Configuração deformada da estrutura (ferramenta).



Fonte: Autor

Nas Figuras 34 e 35, são apresentados os diagramas de momentos fletores da estrutura, gerados pelo Ftool e pela ferramenta desenvolvida neste trabalho, respectivamente.

Figura 34 – Diagrama de momentos fletores (Ftool).



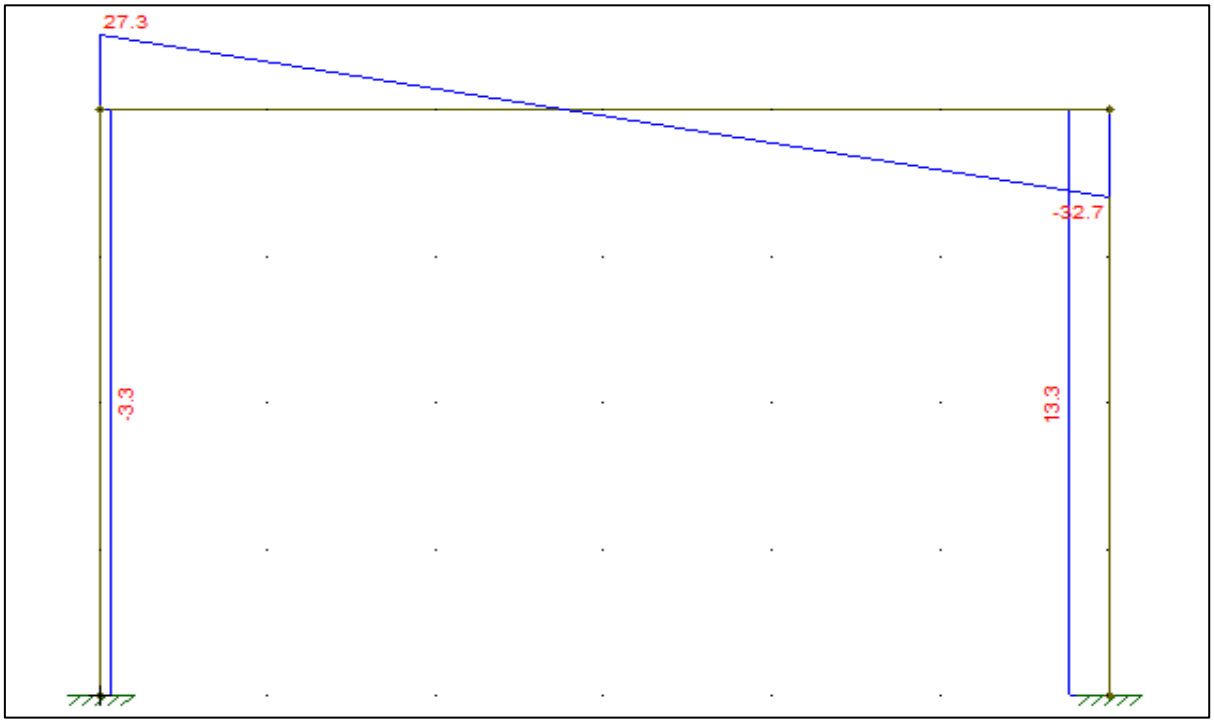
Fonte: Autor

Figura 35 – Diagrama de momentos fletores (ferramenta).

Fonte: Autor

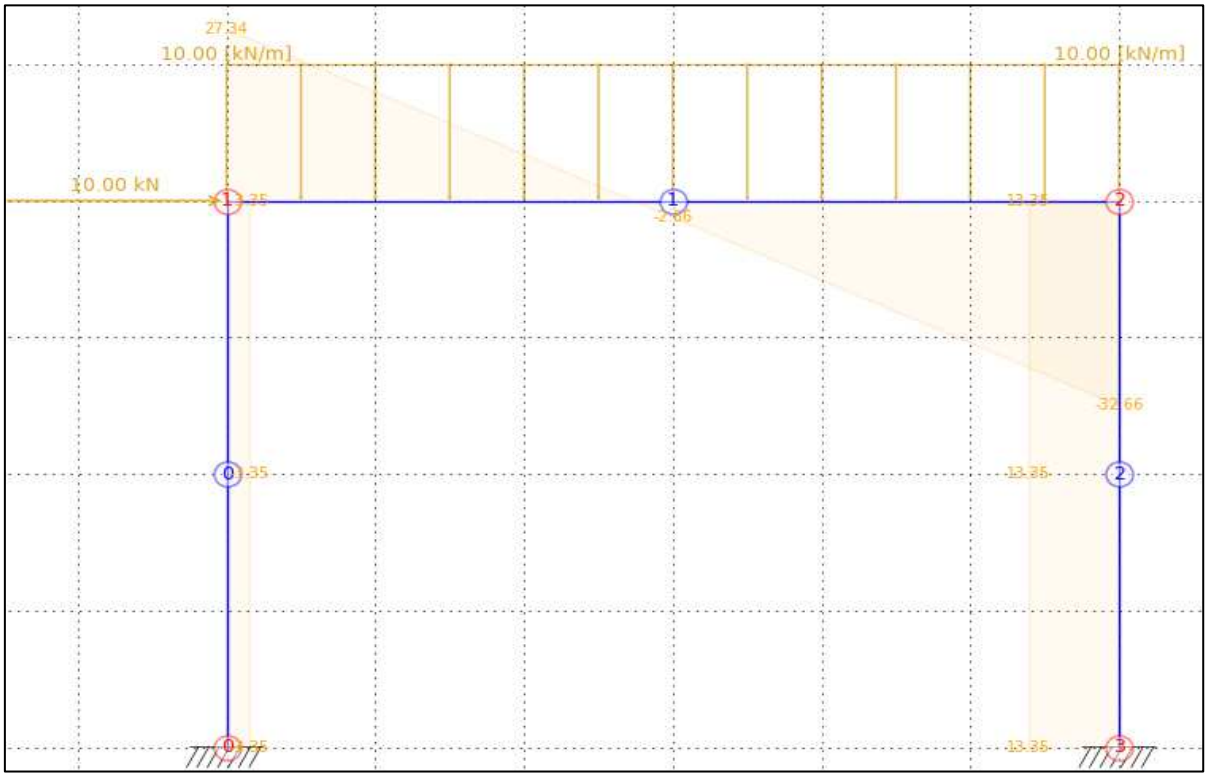
Nas Figuras 36 e 37, são apresentados os diagramas de esforços cortantes da estrutura, gerados pelo Ftool e pela ferramenta desenvolvida neste trabalho, respectivamente.

Figura 36 – Diagrama de esforços cortantes (Ftool).



Fonte: Autor

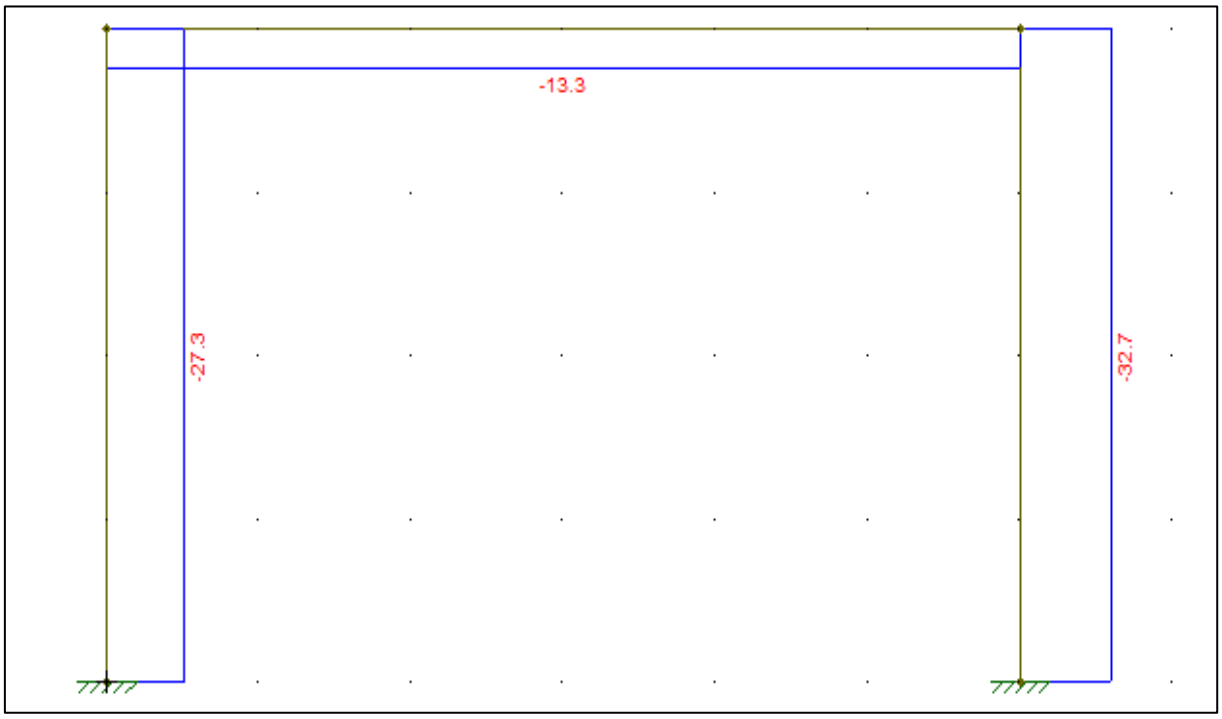
Figura 37 – Diagrama de esforços cortantes (ferramenta).



Fonte: Autor

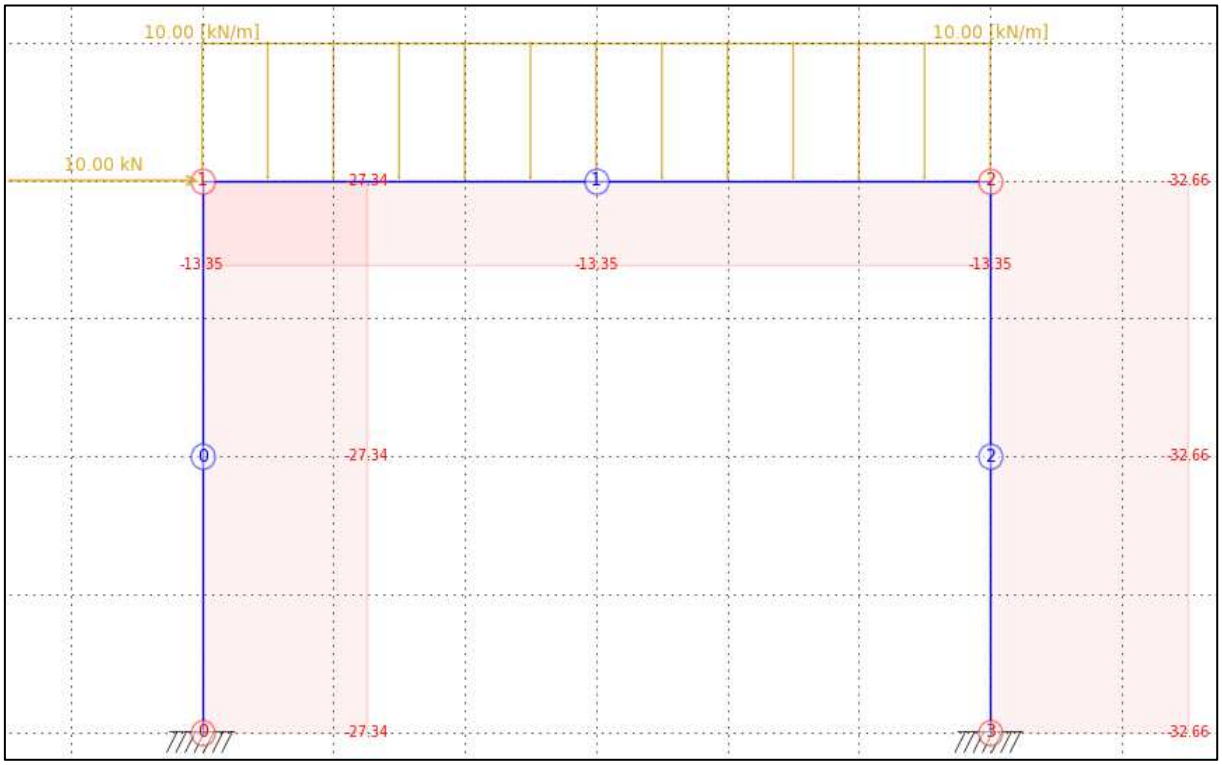
Nas Figuras 38 e 39, são apresentados os diagramas de esforços normais da estrutura, gerados pelo Ftool e pela ferramenta desenvolvida neste trabalho, respectivamente.

Figura 38 – Diagrama de esforços normais (Ftool).



Fonte: Autor

Figura 39 – Diagrama de esforços normais (ferramenta).



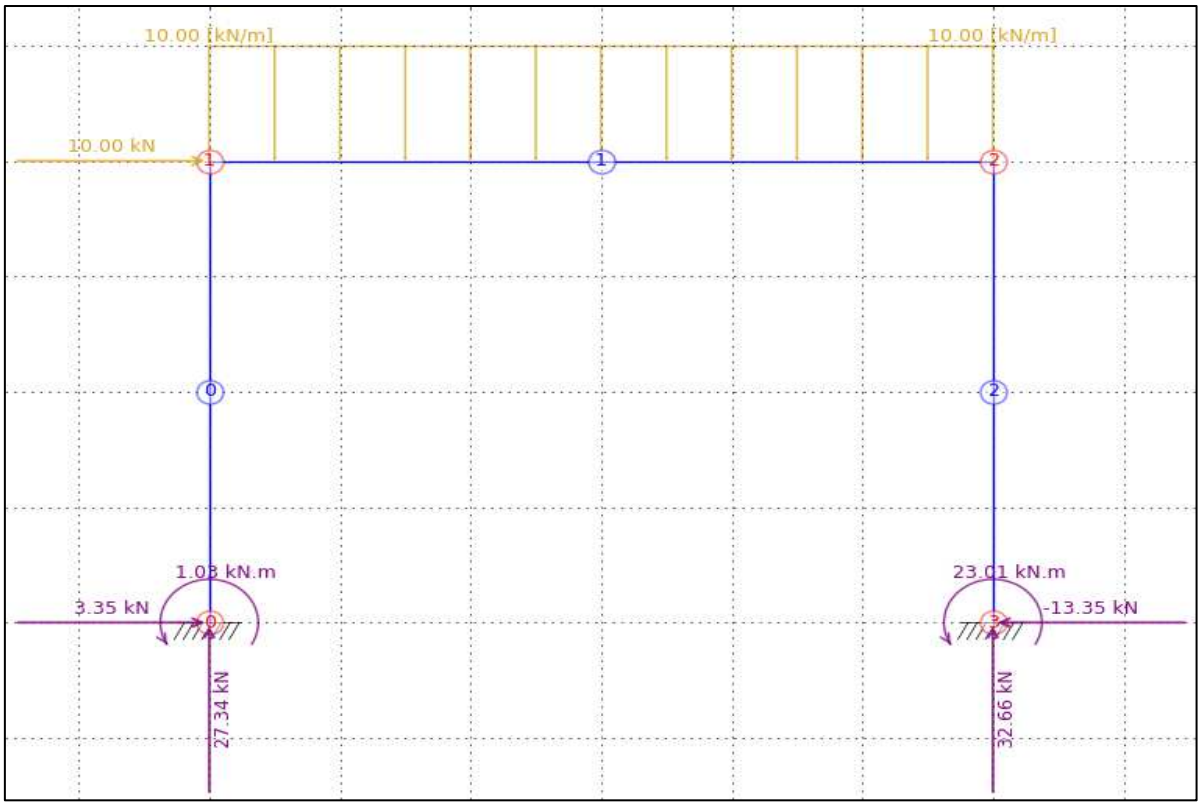
Fonte: Autor

Nas Figuras 40 e 41, são apresentados os resultados das reações nos apoios da estrutura, gerados pelo Ftool e pela ferramenta desenvolvida neste trabalho, respectivamente.

Figura 40 – Resultados das reações de apoio (Ftool).

Fonte: Autor

Figura 41 – Resultados das reações de apoio (ferramenta).



Fonte: Autor

6. CONCLUSÃO

Ao longo deste trabalho, constatou-se que o processo de desenvolvimento de uma ferramenta computacional para análise de estruturas vai muito além do entendimento dos modelos matemáticos e analíticos que servem de base para a sua concepção, bem como do conhecimento da linguagem computacional utilizada em sua codificação. A criação de ferramentas deste gênero também deve compreender conceitos relacionados aos seus aspectos gráficos, de desempenho, de usabilidade, além de outros necessários para tornar a interface gráfica a mais intuitiva possível para os usuários.

Além disso, deve-se compreender que o desenvolvimento de um sistema é um processo iterativo e incremental, que se prolonga por todo o ciclo de vida da ferramenta, seja na forma de manutenções corretivas ou evolutivas, produzindo versões cada vez mais maduras e consistentes ao longo do tempo. Assim, não se pode almejar que a primeira versão de um aplicativo tenha a melhor aparência e o desempenho mais coerente, pois tudo decorre de um processo adaptativo. A ferramenta desenvolvida neste trabalho é um exemplo disso, vez que ainda pode evoluir em diversos aspectos para melhorar a experiência dos usuários.

Apesar disso, levando-se em consideração os objetivos propostos no início deste trabalho e os resultados alcançados ao término desta jornada, pode-se constatar que as metas planejadas preliminarmente foram todas alcançadas, tendo sido implementadas com sucesso as funcionalidades identificadas, dentro das restrições estabelecidas. Ademais, a ferramenta tem se mostrado capaz de atender ao seu propósito principal: realizar a análise de estruturas reticuladas planas.

Embora os objetivos tenham sido alcançados, ainda existem diversas características da ferramenta que podem ser melhoradas, a maioria delas relacionadas a sua usabilidade. Certas particularidades do aplicativo ficaram limitadas ao escopo do projeto, que foi integralmente respeitado em decorrência das restrições de tempo, devido ao prazo para finalização deste trabalho de conclusão de curso. Assim, algumas funcionalidades simples que incrementariam o potencial do aplicativo deixaram de ser adicionadas, tal como a exibição das matrizes de rigidez da estrutura e das barras, que facilitaria a conferência da rotina de cálculos.

Ademais, questões relacionadas ao desempenho também merecem ser revisadas, principalmente quanto ao tratamento de estruturas com elevado número de nós e barras. Quanto maior é número destes elementos, maior é o sistema de equações que a ferramenta deve resolver, tornando o processamento mais lento. No que tange à interface gráfica, a inserção de

nós e barras ficaria mais intuitiva com a utilização de comandos do tipo “arrastar e soltar”, com o auxílio do *mouse*.

Por fim, deve-se ressaltar novamente que o desenvolvimento de uma ferramenta computacional é um processo incremental. Assim, versões futuras deste programa poderão trazer as melhorias desejadas. Contudo, diversos aspectos positivos resultantes deste trabalho merecem ser mencionados, tais como a construção de um código base para ser reutilizado em trabalhos futuros; o desenvolvimento de funcionalidades gráficas para a exibição de diagramas de esforços internos; a criação de um modelo de dados para salvar e carregar estruturas previamente desenhadas; e a elaboração de um relatório de análise de estruturas.

REFERÊNCIAS

AZEVEDO, Álvaro Ferreira Marques. **Método dos Elementos Finitos**. 1. ed. Portugal: Faculdade de Engenharia da Universidade de Porto, 2003.

BATHE, Klays-Jürgen. **Finite Element Procedures**. 2. ed. Watertown, MA: Klaus-Jürgen Bathe, 2014.

BEER, Fernand Pierre; JOHNSTON, E. Russel. **Resistência dos Materiais**. 3. ed. São Paulo: Makron Books, 2006.

HIBBELER, Russell Charles. **Resistência dos Materiais**. 7. ed. São Paulo: Pearson Prentice Hall, 2010.

HIBBELER, Russell Charles. **Análise de Estruturas**. 8. ed. São Paulo: Pearson Education do Brasil, 2013.

LEET, Kenneth M.; Uang, Chia Ming; Gilbert, Anne M. **Fundamentos da Análise Estrutural**. 3. ed. Porto Alegre: AMGH, 2010.

LONGO, Luís Filipe. **Desenvolvimento de um aplicativo de análise de estruturas reticuladas planas em plataforma Android**. Florianópolis: UFSC, 2015.

MARTHA, Luiz Fernando. **Métodos básicos da anaáise de estruturas**. Rio de Janeiro: PUC, 2000.

MARTHA, Luiz Fernando. **O método da rigidez direta sob o enfoque matricial**. Rio de Janeiro: PUC, 1993.

PEREIRA, Carla Soraia da Silva. **Análise de problemas axiais planos:** implementação computacional utilizando o método de elementos finitos e linguagem C. Caruaru: UFPE, 2011.

RIBEIRO, Fernando Luiz Bastos. **Introdução ao Método dos Elementos Finitos**. Rio de Janeiro: UFRJ, 2004.

SÜSSEKIND, José Carlos. **Curso de Análise Estrutural – Volume 1:** Estruturas Isostáticas. Porto Alegre: Globo, 1981.

SÜSSEKIND, José Carlos. **Curso de Análise Estrutural – Volume 2:** Deformações em Estruturas e Método das Forças. Porto Alegre: Globo, 1980.

SÜSSEKIND, José Carlos. **Curso de Análise Estrutural – Volume 3:** Método das deformações e Processo de Cross. Rio de Janeiro: Globo, 1987.

SORIANO, Humberto Lima. **Análise de Estruturas:** método das forças e método dos deslocamentos. 1a. ed, Editora Ciência Moderna, Rio de Janeiro - RJ, 2004.

ZIENKIEWICZ, Olgierd C.; TAYLOR, Robert L. **The finite element method – Volume 1:** The basis. 5. ed. Oxford: Butterworth-Heinemann, 2000.

KASSIMALI, Aslam. **Matrix Analysis of Structures**. 2. ed. Stamford, CT: Cengage Learning, 2012.

CRONOGRAMA DE ATIVIDADES

ATIVIDADES	NOV/2016	DEZ/2016	JAN/2017	FEV/2017	MAR/2017	ABR/2017	MAI/2017	JUN/2017
1. Elaboração do Projeto								
2. Fichamento								
3. Entrega do Projeto								
4. Pesquisa Bibliográfica								
5. Desenvolvimento da Ferramenta								
6. Elaboração do TCC								
7. Conclusão								
8. Entrega do TCC								
9. Defesa do TCC								

APÊNDICE A – CÓDIGOS-FONTE DOS COMPONENTES ESTRUTURAIS

A.1 Classe Nó:

```
1. import math
2. from componentes.estrutura.cargaresultante import CargaConcentradaResultante
3. 
4. class No(object):
5. 
6.     def __init__(self, x, y):
7.         self.x = x
8.         self.y = y
9.         self.condicao_apoio = [1,1,1]
10.         self.carga_resultante = CargaConcentradaResultante(0,0,0)
11.         self.cargas_concentradas = []
12. 
13.     def __eq__(self, other) :
14.         if math.fabs(self.x - other.x) < 0.01 and math.fabs(self.y - other.y) < 0.01:
15.             return True
16.         else:
17.             return False
```

A.2 Classe Seção:

```
1. class Secao(object):
2.     def __init__(self, A, I):
3.         self.A = A
4.         self.I = I
5. 
6.     def __eq__(self, other) :
7.         if self.A == other.A and self.I == other.I:
8.             return True
9.         else:
10.             return False
11. 
12.     def __str__(self):
13.         return '%f %f' % (self.A, self.I)
```

A.3 Classe Material:

```
1. class Material(object):
2.     def __init__(self, E):
3.         self.E = E
4. 
5.     def __eq__(self, other) :
6.         if self.E == other.E:
7.             return True
8.         else:
9.             return False
10. 
11.     def __str__(self):
12.         return '%f' % (self.E)
```

A.4 Classe Barra

```
1.  import math
2.  import numpy as np
3.  from componentes.estrutura.cargaresultante import CargaDistribuidaResultan
4. 
5.  class Barra(object):
```

```
6.      def __init__(self, no_1, no_2, material, secao):
7.          self.no_1 = no_1
8.          self.no_2 = no_2
9.          self.material = material
10.         self.secao = secao
11.         self.comprimento = self.calcular_comprimento()
12.         self.cargas_distribuidas = []
13.         self.carga_resultante = CargaDistribuidaResultante(0,0,0,0,0,0)
14.         self.condicao_rotulas = [1,1]
15.
16.     def calcular_comprimento(self):
17.         return np.sqrt( np.power( self.no_1.x - self.no_2.x, 2 ) +  np.power( self.n
    o_1.y - self.no_2.y, 2 ) )
18.
19.     def gerar_matriz_rigidez_local(self):
20.
21.         E = self.material.E
22.         A = self.secao.A
23.         I = self.secao.I
24.
25.         x_1 = self.no_1.x
26.         y_1 = self.no_1.y
27.         x_2 = self.no_2.x
28.         y_2 = self.no_2.y
29.
30.         L = math.sqrt((x_2-x_1)**2+(y_2-y_1)**2)
31.
32.         EA = E*A
33.         EI = E*I
34.         EA_L = EA/L
35.         EI_L = EI/L
36.         EI_L2 = EI_L/L
37.         EI_L3 = EI_L2/L
38.
39.         Z = 0.000001
40.
41.         if self.condicao_rotulas[0] == 0 and self.condicao_rotulas[1] == 0:
42.                 # barra biarticulada
43.                 kl=[[      EA_L,           Z,           Z,       -
    EA_L,           Z,           Z  ],
44.                     [         Z,           Z,           Z,           Z,           Z,
                Z  ],
45.                     [         Z,           Z,           Z,           Z,           Z,
                Z  ],
46.                     [     -
    EA_L,           Z,           Z,        EA_L,           Z,           Z  ],
47.                     [         Z,           Z,           Z,           Z,           Z,
                Z  ],
48.                     [         Z,           Z,           Z,           Z,           Z,
                Z  ]]
49.         elif self.condicao_rotulas[0] == 1 and self.condicao_rotulas[1] == 0 :
50.                 # barra engastada-articulada
51.                 kl=[[      EA_L,           Z,           Z,       -
    EA_L,           Z,           Z  ],
52.                     [         Z,     3*EI_L3,     3*EI_L2,           Z,    -
    3*EI_L3,           Z  ],
53.                     [         Z,     3*EI_L2,      3*EI_L,           Z,    -
    3*EI_L2,           Z  ],
54.                     [     -
    EA_L,           Z,           Z,        EA_L,           Z,           Z  ],
55.                     [         Z,    -3*EI_L3,    -
    3*EI_L2,           Z,     3*EI_L3,           Z  ],
56.                     [         Z,           Z,           Z,           Z,           Z,
                Z  ]]
57.         elif self.condicao_rotulas[0] == 0 and self.condicao_rotulas[1] == 1 :
58.                 # barra articulada-engastada
```

```
                kl=[[      EA_L,           Z,           Z,       -
    EA_L,           Z,           Z  ],
                    [         Z,     3*EI_L3,           Z,           Z,    -
    3*EI_L3,      3*EI_L2  ],
                    [         Z,           Z,           Z,           Z,           Z,
              Z  ],
                    [     -
    EA_L,           Z,           Z,        EA_L,           Z,           Z  ],
                    [         Z,    -
    3*EI_L3,           Z,           Z,     3*EI_L3,     -3*EI_L2  ],
                    [         Z,     3*EI_L2,           Z,           Z,    -
    3*EI_L2,       3*EI_L  ]]
        elif self.condicao_rotulas[0] == 1 and self.condicao_rotulas[1] == 1 :
                # barra articulada-engastada
                kl=[[      EA_L,           Z,           Z,       -
    EA_L,           Z,           Z  ],
                    [         Z,    12*EI_L3,     6*EI_L2,           Z,   -
    12*EI_L3,      6*EI_L2  ],
                    [         Z,     6*EI_L2,      4*EI_L,           Z,    -
    6*EI_L2,       2*EI_L  ],
                    [     -
    EA_L,           Z,           Z,        EA_L,           Z,           Z  ],
                    [         Z,   -12*EI_L3,    -
    6*EI_L2,           Z,    12*EI_L3,     -6*EI_L2  ],
                    [         Z,     6*EI_L2,      2*EI_L,           Z,    -
    6*EI_L2,       4*EI_L  ]]
        return np.matrix(kl)

    def gerar_matriz_transformacao(self):

        x_1 = self.no_1.x
        y_1 = self.no_1.y
        x_2 = self.no_2.x
        y_2 = self.no_2.y

        theta = math.atan2(y_2 - y_1, x_2 - x_1)

        sin_theta = math.sin(theta)
        cos_theta = math.cos(theta)

        T = [[  cos_theta,   -
    sin_theta,           0,           0,           0,           0  ],
             [  sin_theta,    cos_theta,           0,           0,           0,
          0  ],
             [          0,           0,           1,           0,           0,
          0  ],
             [          0,           0,           0,    cos_theta,   -
    sin_theta,           0  ],
             [          0,           0,           0,    sin_theta,    cos_theta,
          0  ],
             [          0,           0,           0,           0,           0,
          1  ]]

        return np.matrix(T)

    def gerar_matriz_g(self):

        condicao_apoio_junta_1 = self.no_1.condicao_apoio
        condicao_apoio_junta_2 = self.no_2.condicao_apoio

        g = np.concatenate((condicao_apoio_junta_1, condicao_apoio_junta_2))

        return np.array(g)

    def gerar_matriz_kg(self, T, kl):
        return np.matrix(T * kl * np.transpose(T))
```

```
107.
108.     def gerar_matriz_carga_local(self):
109.         return np.transpose(np.matrix([self.carga_resultante.vetor()]))
110.
111.     def gerar_matriz_carga_global(self, T, fl):
112.         return T*fl
113.
114.     def __str__(self):
115.         return '[%s; %s; %s; %s]' % (str(self.no_1), str(self.no_2), str(self.materi
    al), str(self.secao))
```

A.5 Classe Carga Concentrada

```
1. #CargaConcentrada.py
2. import math
3.
4. class CargaConcentrada(object):
5.
6.     def __init__(self, no, fx, fy, mz):
7.         self.fx = fx
8.         self.fy = fy
9.         self.mz = mz
10.        self.no = no
```

A.6 Classe Carga Distribuída

```
1. #CargaDistribuida.py
2. import math
3.
4. from componentes.estrutura.cargaresultante import CargaDistribuidaResultante
5.
6. class CargaDistribuida(object):
7.
8.     def __init__(self, barra, qx_1, qx_2, qy_1, qy_2):
9.         self.qx_1 = qx_1
10.        self.qx_2 = qx_2
11.        self.qy_1 = qy_1
12.        self.qy_2 = qy_2
13.        self.barra = barra
14.
15.    def computar_esforcos_engastamento_perfeito(self):
16.
17.        qx_1 = self.qx_1
18.        qx_2 = self.qx_2
19.        qy_1 = self.qy_1
20.        qy_2 = self.qy_2
21.        barra = self.barra
22.        L = self.barra.comprimento
23.
24.        if barra.condicao_rotulas[0] == 0 and barra.condicao_rotulas[1] == 0:
25.                # barra biarticulada
26.                return CargaDistribuidaResultante( (-1/6)*(2*qx_1+qx_2)*L, (-
    1/6)*(2*qy_1+qy_2)*L, 0, (-1/6)*(qx_1+2*qx_2), (-1/6)*(qy_1+2*qy_2)*L, 0 )
27.        elif barra.condicao_rotulas[0] == 1 and barra.condicao_rotulas[1] == 0 :
28.                # barra engastada-articulada
29.                return CargaDistribuidaResultante( (-1/6)*(2*qx_1+qx_2)*L, (-
    1/120)*(48*qy_1+27*qy_2)*L,  (-1/120)*(8*qy_1+7*qy_2)*L*L, (-1/6)*(qx_1+2*qx_2), (-
    1/120)*(12*qy_1+33*qy_2)*L, 0 )
30.        elif barra.condicao_rotulas[0] == 0 and barra.condicao_rotulas[1] == 1 :
31.                # barra articulada-engastada
32.                return CargaDistribuidaResultante( (-1/6)*(2*qx_1+qx_2)*L, (-
    1/120)*(33*qy_1+12*qy_2)*L, 0, (-1/6)*(qx_1+2*qx_2), (-
    1/120)*(27*qy_1+48*qy_2)*L, (+1/120)*(7*qy_1+8*qy_2)*L*L )
33.        elif barra.condicao_rotulas[0] == 1 and barra.condicao_rotulas[1] == 1 :
```

```
                # barra engastada-engastada
                return CargaDistribuidaResultante( (-1/6)*(2*qx_1+qx_2)*L, (-
1/20)*(7*qy_1+3*qy_2)*L, (-1/120)*(6*qy_1+4*qy_2)*L*L, (-1/6)*(qx_1+2*qx_2), (-
1/20)*(3*qy_1+7*qy_2)*L, (+1/120)*(4*qy_1+6*qy_2)*L*L )
        else:
                return CargaDistribuidaResultante( 0, 0, 0, 0, 0, 0 )
```

A.7 Classe Carga Resultante

```
#CargaResultante.py

class CargaConcentradaResultante(object):

    def __init__(self, fx, fy, mz):
        self.fx = fx
        self.fy = fy
        self.mz = mz

    def indice(self, i):
        if i == 0:
            return self.fx
        elif i == 1:
            return self.fy
        elif i == 2:
            return self.mz

    def vetor(self):
        return [self.fx, self.fy, self.mz]

class CargaDistribuidaResultante(object):

    def __init__(self, fx1, fy1, mz1, fx2, fy2, mz2):
        self.fx1 = fx1
        self.fy1 = fy1
        self.mz1 = mz1
        self.fx2 = fx2
        self.fy2 = fy2
        self.mz2 = mz2

    def indice(self, i):
        if i == 0:
            return self.fx1
        elif i == 1:
            return self.fy1
        elif i == 2:
            return self.mz1
        elif i == 3:
            return self.mx2
        elif i == 4:
            return self.my2
        elif i == 5:
            return self.mz2

    def vetor(self):
        return [self.fx1, self.fy1, self.mz1, self.fx2, self.fy2, self.mz2]
```

A.8 Classe Estrutura

```
#Estrutura.py

import math
import numpy as np

from .no import No
```

```
7.  from .barra import Barra
8.  from .cargaconcentrada import CargaConcentrada
9.  from .cargadistribuida import CargaDistribuida
10. from .material import Material
11. from .secao import Secao
12.
13. class Estrutura(object):
14.     def __init__(self):
15.         self.numero_nos_por_barra = 2
16.         self.grau_liberdade_por_no = 3
17.         self.grau_liberdade_por_barra = self.numero_nos_por_barra * self.grau_liberd
    ade_por_no
18.         self.nos = []
19.         self.subnos_discretizadas = {}
20.         self.barras = []
21.         self.subnos_barras_discretizados = {}
22.         self.subbarras_barras_discretizados = {}
23.         self.materiais = []
24.         self.secoes = []
25. #         self.cargas_distribuidas = []
26. #         self.cargas_concentradas = []
27.
28.     def discretizar(self):
29.
30.         estrutura = Estrutura()
31.
32.         for barra in self.barras:
33.
34.             L = barra.comprimento
35.             material = barra.material
36.             secao = barra.secao
37.             no1 = barra.no_1
38.             no5 = barra.no_2
39.
40.             x1 = no1.x
41.             y1 = no1.y
42.             x5 = no5.x
43.             y5 = no5.y
44.
45.             x2 = (1-(1/6)) * x1 + (1/6) * x5
46.             y2 = (1-(1/6)) * y1 + (1/6) * y5
47.             x3 = (1-(3/6)) * x1 + (3/6) * x5
48.             y3 = (1-(3/6)) * y1 + (3/6) * y5
49.             x4 = (1-(5/6)) * x1 + (5/6) * x5
50.             y4 = (1-(5/6)) * y1 + (5/6) * y5
51.
52.             idc_material = estrutura.adicionar_material(material.E)
53.             idc_secao = estrutura.adicionar_secao(secao.A,secao.I)
54.
55.             idc_no1 = estrutura.adicionar_no(x1,y1)
56.             idc_no2 = estrutura.adicionar_no(x2,y2)
57.             idc_no3 = estrutura.adicionar_no(x3,y3)
58.             idc_no4 = estrutura.adicionar_no(x4,y4)
59.             idc_no5 = estrutura.adicionar_no(x5,y5)
60.
61.             idc_barra1 = estrutura.adicionar_barra(idc_no1,idc_no2,idc_material,idc_
    secao)
62.             idc_barra2 = estrutura.adicionar_barra(idc_no2,idc_no3,idc_material,idc_
    secao)
63.             idc_barra3 = estrutura.adicionar_barra(idc_no3,idc_no4,idc_material,idc_
    secao)
64.             idc_barra4 = estrutura.adicionar_barra(idc_no4,idc_no5,idc_material,idc_
    secao)
65.
66.             idc_barra = self.barras.index(barra)
67.             idc_no_1 = self.nos.index(barra.no_1)
```

```
68.             idc_no_2 = self.nos.index(barra.no_2)
69.
70.             estrutura.subnos_barras_discretizados[idc_barra] = [idc_no1, idc_no2, id
    c_no3, idc_no4, idc_no5]
71.             estrutura.subbarras_barras_discretizados[idc_barra] = [idc_barra1, idc_b
    arra2, idc_barra3, idc_barra4]
72.             estrutura.subnos_discretizadas[idc_no_1] = idc_no1
73.             estrutura.subnos_discretizadas[idc_no_2] = idc_no5
74.
75.             estrutura.especificar_condicao_rotulas(idc_barra1,barra.condicao_rotulas
    [0],1)
76.             estrutura.especificar_condicao_rotulas(idc_barra2,1,1)
77.             estrutura.especificar_condicao_rotulas(idc_barra3,1,1)
78.             estrutura.especificar_condicao_rotulas(idc_barra4,1,barra.condicao_rotul
    as[1])
79.
80.             for carga_distribuida in barra.cargas_distribuidas:
81.                 qx_1 = carga_distribuida.qx_1
82.                 qx_2 = carga_distribuida.qx_2
83.                 qy_1 = carga_distribuida.qy_1
84.                 qy_2 = carga_distribuida.qy_2
85.
86.                 qy_s_1 = qy_1 + (qy_2-qy_1)*(1/6)
87.                 qy_s_2 = qy_1 + (qy_2-qy_1)*(3/6)
88.                 qy_s_3 = qy_1 + (qy_2-qy_1)*(5/6)
89.                 qx_s_1 = qx_1 + (qx_2-qx_1)*(1/6)
90.                 qx_s_2 = qx_1 + (qx_2-qx_1)*(3/6)
91.                 qx_s_3 = qx_1 + (qx_2-qx_1)*(5/6)
92.
93.                 estrutura.adicionar_carga_distribuida(idc_barra1,   qx_1,  qx_s_1,
      qy_1,  qy_s_1)
94.                 estrutura.adicionar_carga_distribuida(idc_barra2, qx_s_1,  qx_s_2,
    qy_s_1,  qy_s_2)
95.                 estrutura.adicionar_carga_distribuida(idc_barra3, qx_s_2,  qx_s_3,
    qy_s_2,  qy_s_3)
96.                 estrutura.adicionar_carga_distribuida(idc_barra4, qx_s_3,    qx_2,
    qy_s_3,    qy_2)
97.
98.         for idc_no, no in enumerate(self.nos):
99.             idc_no_discretizada = estrutura.subnos_discretizadas[idc_no]
100.            estrutura.adicionar_carga_concentrada(idc_no_discretizada, no.carga_resu
    ltante.fx, no.carga_resultante.fy, no.carga_resultante.mz)
101.            estrutura.especificar_condicao_contorno(idc_no_discretizada,no.condicao_
    apoio[0],no.condicao_apoio[1],no.condicao_apoio[2])
102.
103.        return estrutura
104.
105.    def adicionar_material(self, E):
106.        material = Material(E)
107.
108.        try:
109.            idc_material = self.materiais.index(material)
110.        except ValueError:
111.            self.materiais.append(material)
112.            idc_material = self.materiais.index(material)
113.
114.        return idc_material + 1
115.
116.    def adicionar_secao(self, A, I):
117.        secao = Secao(A, I)
118.
119.        try:
120.            idc_secao = self.secoes.index(secao)
121.        except ValueError:
122.            self.secoes.append(secao)
123.            idc_secao = self.secoes.index(secao)
```

```
124.
125.         return idc_secao + 1
126.
127.     def adicionar_no(self, x, y):
128.         no = No(x, y)
129.
130.         try:
131.             idc_no = self.nos.index(no)
132.         except ValueError:
133.             self.nos.append(no)
134.             idc_no = self.nos.index(no)
135.
136.         return idc_no + 1
137.
138.     def adicionar_barra(self, i, j, idcMaterial, idcSecao):
139.         no_1 = self.nos[i-1]
140.         no_2 = self.nos[j-1]
141.         material = self.materiais[idcMaterial-1]
142.         secao = self.secoes[idcSecao-1]
143.         barra = Barra( no_1, no_2, material, secao )
144.
145.         try:
146.             idc_barra = self.barras.index(barra)
147.         except ValueError:
148.             self.barras.append(barra)
149.             idc_barra = self.barras.index(barra)
150.
151.         return idc_barra + 1
152.
153.     def especificar_condicao_contorno(self, indice_no, deslocamento_x, deslocamento_
    y, rotacao_z):
154.         self.nos[indice_no-
    1].condicao_apoio = [deslocamento_x, deslocamento_y, rotacao_z]
155.
156.     def especificar_condicao_rotulas(self, indice_barra, rotula_1, rotula_2):
157.         self.barras[indice_barra-1].condicao_rotulas = [rotula_1, rotula_2]
158.
159.     def adicionar_carga_concentrada(self, indice_no, fx, fy, mz):
160.
161.         no = self.nos[indice_no-1]
162.
163.         carga_concentrada = CargaConcentrada(no, fx, fy, mz)
164.
165.         no.cargas_concentradas.append(carga_concentrada)
166.
167.         no.carga_resultante.fx += fx
168.         no.carga_resultante.fy += fy
169.         no.carga_resultante.mz += mz
170.
171.     def adicionar_carga_distribuida(self, indice_barra, qx_1, qx_2, qy_1, qy_2):
172.
173.         barra = self.barras[indice_barra-1]
174.
175.         carga_distribuida = CargaDistribuida(barra, qx_1, qx_2, qy_1, qy_2)
176.
177.         barra.cargas_distribuidas.append(carga_distribuida)
178.
179.         resultante = carga_distribuida.computar_esforcos_engastamento_perfeito()
180.
181.         barra.carga_resultante.fx1 += resultante.fx1
182.         barra.carga_resultante.fy1 += resultante.fy1
183.         barra.carga_resultante.mz1 += resultante.mz1
184.         barra.carga_resultante.fx2 += resultante.fx2
185.         barra.carga_resultante.fy2 += resultante.fy2
186.         barra.carga_resultante.mz2 += resultante.mz2
187.
```

```
188.     def calcular(self):
189.
190.         n = 0
191.
192.         for i in range(1, len(self.nos)+1):
193.             no = self.nos[i-1]
194.             for j in range(1, len(no.condicao_apoio)+1):
195.                 if no.condicao_apoio[j-1] != 0 :
196.                     n += 1
197.                     no.condicao_apoio[j-1] = n
198.
199.         F = np.zeros((n,1))
200.
201.         F = self.montar_carga_no(F)
202.
203.         K = np.zeros((n,n))
204.
205.         print (F)
206.
207.         for barra in self.barras :
208.             T = barra.gerar_matriz_transformacao()
209.             kl = barra.gerar_matriz_rigidez_local()
210.             kg = barra.gerar_matriz_kg(T,kl)
211.             fl = barra.gerar_matriz_carga_local()
212.             fg = barra.gerar_matriz_carga_global(T,fl)
213.             g = barra.gerar_matriz_g()
214.             K = self.gerar_matriz_rigidez_global(K,kg,g)
215.             F = self.montar_carga_barra(F,fg,g)
216.
217.         delta = np.linalg.solve(K,F)
218.         no_disp = np.zeros((len(self.nos),3))
219.
220.         for i in range(1, len(self.nos)+1):
221.             no = self.nos[i-1]
222.             for j in range(1,len(no.condicao_apoio)+1):
223.                 if no.condicao_apoio[j-1] != 0:
224.                     no_disp[i-1,j-1] = delta[no.condicao_apoio[j-1]-1]
225.
226.         forca_l = np.zeros((len(self.barras),6))
227.         forca_g = np.zeros((len(self.barras),6))
228.
229.         for i in range(1, len(self.barras)+1):
230.             barra = self.barras[i-1]
231.             kl = barra.gerar_matriz_rigidez_local()
232.             T = barra.gerar_matriz_transformacao()
233.             kg = barra.gerar_matriz_kg(T,kl)
234.             g = barra.gerar_matriz_g()
235.
236.             vdegv = np.zeros((1,self.grau_liberdade_por_barra))
237.             for j in range(1, self.grau_liberdade_por_barra+1):
238.                 if g[j-1] != 0:
239.                     vdegv[0,j-1] = delta[g[j-1]-1]
240.
241.             fg = kg*np.transpose(vdegv)
242.             fl = np.transpose(T)*fg
243.             f0 = barra.gerar_matriz_carga_local()
244.
245.             forca_l[i-1,:]=(np.transpose(fl-f0))[0,:]
246.             forca_g[i-1,:]=(np.transpose(T*(fl-f0)))[0,:]
247.
248.         self.F = F
249.         self.delta = delta
250.         self.no_disp = no_disp
251.         self.forca_l = forca_l
252.         self.forca_g = forca_g
253.
```

```
254.     def gerar_modelo(self):
255.
256.         modelo = ""
257.
258.         modelo += '#materiais (Rótulo, E)'.format(i, material.E)
259.         for i, material in enumerate(self.materiais):
260.             modelo += '{:>10} {:>10}\n'.format(i, material.E)
261.
262.         modelo += '#secoes (Rótulo, A, I)'.format(i, material.E)
263.         for i, secao in enumerate(self.secoes):
264.             modelo += '{:>10} {:>10} {:>10}\n'.format(i, secao.A, secao.I)
265.
266.         modelo += '#nos (X, Y)'.format(i, material.E)
267.         for i, no in enumerate(self.nos):
268.             modelo += '{:>10} {:>10} {:>10}\n'.format(i, no.x, no.y)
269.
270.         modelo += '#barras (Índice no inicial, Índice no final)'.format(i, material.
    E)
271.         for i, barra in enumerate(self.barras):
272.             modelo += '{:>10}, {:>10}, {:>10}, {:>10}, {:>10}\n'.format(i, self.nos.
    index(barra.no_1), self.nos.index(barra.no_1), self.materiais.index(barra.material),
     self.secoes.index(barra.secao) )
273.
274.         return modelo
275.
276.     def carregar_modelo(self, modelo):
277.
278.         modelo = ""
279.
280.         return modelo
281.
282.     def exportar_resultados(self, filename):
283.
284.         arquivo = open(filename, 'w')
285.
286.         arquivo.write('Imprimindo os resultados da análise estrutural:\n')
287.         arquivo.write('\n')
288.
289.         arquivo.write('1. Vetor global de forças (F):\n')
290.         arquivo.write('\n')
291.         arquivo.write('+-----------+\n')
292.         arquivo.write('|     F     |\n')
293.         arquivo.write('+-----------+\n')
294.         for i in range(1, self.F.shape[0]+1):
295.             arquivo.write('|')
296.             for j in range(1, self.F.shape[1]+1):
297.                 arquivo.write('{:11.2f}|'.format(self.F[i-1,j-1]))
298.             arquivo.write('\n')
299.         arquivo.write('+-----------+\n')
300.         arquivo.write('\n')
301.
302.         arquivo.write('2. Vetor solução de deslocamentos nodais (Delta):\n')
303.         arquivo.write('\n')
304.         arquivo.write('+---------------+\n')
305.         arquivo.write('|     Delta     |\n')
306.         arquivo.write('+---------------+\n')
307.         for i in range(1, self.delta.shape[0]+1):
308.             arquivo.write('|')
309.             for j in range(1, self.delta.shape[1]+1):
310.                 arquivo.write('{:15.8f}|'.format(self.delta[i-1,j-1]))
311.             arquivo.write('\n')
312.         arquivo.write('+---------------+\n')
313.         arquivo.write('\n')
314.
315.         arquivo.write('3. Deslocamentos nodais:\n')
316.         arquivo.write('\n')
```

```
317.         arquivo.write('+----+----------------+----------------+----------------
    +\n')
318.         arquivo.write('| No | Deslocamento X | Deslocamento Y |    Rotação Z   |\n')
319.         arquivo.write('+----+----------------+----------------+----------------
    +\n')
320.         for i in range(1, self.no_disp.shape[0]+1):
321.             arquivo.write('|{:4}|'.format(i))
322.             for j in range(1, self.no_disp.shape[1]+1):
323.                 arquivo.write('{:16.8f}|'.format(self.no_disp[i-1,j-1]))
324.             arquivo.write('\n')
325.         arquivo.write('+----+----------------+----------------+----------------
    +\n')
326.         arquivo.write('\n')
327.
328.         arquivo.write('4. Ações nas barras em coordenadas locais:\n')
329.         arquivo.write('\n')
330.         arquivo.write('+-------+-----------+-----------+-----------+-----------+----
    -------+-----------+\n')
331.         arquivo.write('| Barra |    Fx1    |    Fy1    |    Mz1    |    Fx2    |
    Fy2    |    Mz2    |\n')
332.         arquivo.write('+-------+-----------+-----------+-----------+-----------+----
    -------+-----------+\n')
333.         for i in range(1, self.forca_l.shape[0]+1):
334.             arquivo.write('|{:6} |'.format(i))
335.             for j in range(1, self.forca_l.shape[1]+1):
336.                 arquivo.write('{:11.4f}|'.format(self.forca_l[i-1,j-1]))
337.             arquivo.write('\n')
338.         arquivo.write('+-------+-----------+-----------+-----------+-----------+----
    -------+-----------+\n')
339.         arquivo.write('\n')
340.
341.         arquivo.write('5. Ações nas barras em coordenadas globais:\n')
342.         arquivo.write('\n')
343.         arquivo.write('+-------+-----------+-----------+-----------+-----------+----
    -------+-----------+\n')
344.         arquivo.write('| Barra |    Fx1    |    Fy1    |    Mz1    |    Fx2    |
    Fy2    |    Mz2    |\n')
345.         arquivo.write('+-------+-----------+-----------+-----------+-----------+----
    -------+-----------+\n')
346.         for i in range(1, self.forca_g.shape[0]+1):
347.             arquivo.write('|{:6} |'.format(i))
348.             for j in range(1, self.forca_g.shape[1]+1):
349.                 arquivo.write('{:11.4f}|'.format(self.forca_g[i-1,j-1]))
350.             arquivo.write('\n')
351.         arquivo.write('+-------+-----------+-----------+-----------+-----------+----
    -------+-----------+\n')
352.         arquivo.close()
353.
354.     def gerar_matriz_rigidez_global(self, K, kg, g):
355.         for i in range(1, self.grau_liberdade_por_barra+1):
356.             if g[i-1] != 0:
357.                 for j in range(1, self.grau_liberdade_por_barra+1):
358.                     if g[j-1] != 0:
359.                         K[g[i-1]-1, g[j-1]-1] = K[g[i-1]-1, g[j-1]-1] + kg[i-1, j-
    1]
360.         return K
361.
362.     def montar_carga_no(self, F):
363.         for i in range(1,len(self.nos)+1):
364.             no = self.nos[i-1]
365.             for j in range(1,len(no.condicao_apoio)+1):
366.                 if no.condicao_apoio[j-1] != 0:
367.                     F[no.condicao_apoio[j-1]-1] = no.carga_resultante.indice(j-1)
368.         return F
369.
```

```
370.     def montar_carga_barra(self, F, fg, g):
371.         for i in range(1, self.grau_liberdade_por_barra+1):
372.             if g[i-1] != 0:
373.                 F[g[i-1]-1]=F[g[i-1]-1]+fg[i-1]
374.         return F
375.
376.     def imprimir_nos(self):
377.         print('{:>10} {:>10} {:>10}'.format('X', 'Y', 'Apoio'))
378.         for no in self.nos:
379.             print('{:>10} {:>10} {:>10}'.format(no.x, no.y, '0'))
380.
381.     def imprimir_barras(self):
382.         for barra in self.barras:
383.             print(barra)
384.
385.     def imprimir_cargas(self):
386.         for barra in self.barras:
387.             print(barra.carga_resultante.vetor())
```

APÊNDICE B – CÓDIGOS-FONTE DOS COMPONENTES DE INTERFACE GRÁFICA

B.1 Classe Canvas (Área de Desenho)

```
1. From __future__ import unicode_literals
2. import sys
3. import os
4. import random
5. import matplotlib
6. import math
7.
8. matplotlib.use('Qt5Agg')
9. from PyQt5 import QtCore, QtGui, QtWidgets
10.
11.         from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as Naviga
tionToolbar
12.         from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCan
vas
13.         from matplotlib.figure import Figure
14.         from matplotlib.ticker import MultipleLocator
15.         from matplotlib.lines import Line2D
16.         from matplotlib.patches import Arc, RegularPolygon
17.         from matplotlib.path import Path
18.         from matplotlib.collections import PatchCollection
19.         from matplotlib.transforms import offset_copy
20.         import matplotlib.patches as patches
21.         import matplotlib.pyplot as plt
22.         import numpy as np
23.         from numpy import radians as rad
24.         from scipy import interpolate
25.
26.         class MyMplCanvas(FigureCanvas):
27.
28.             def __init__(self, parent=None, width=10, height=4, dpi=100):
29.                 self.fig = Figure(figsize=(width, height), dpi=dpi)
30.
31.                 self.axes = self.fig.add_subplot(111)
32.                 self.fig.set_tight_layout(True)
33.
34.                 self.compute_initial_figure()
35.
36.                 FigureCanvas.__init__(self, self.fig)
37.                 FigureCanvas.setSizePolicy(self, QtWidgets.QSizePolicy.Expanding, QtW
idgets.QSizePolicy.Expanding)
38.                 FigureCanvas.updateGeometry(self)
39.                 self.setParent(parent)
40.
41.             def limpar(self):
42.                 self.axes.clear()
43.                 self.compute_initial_figure()
44.                 self.draw()
45.
46.             def compute_initial_figure(self):
47.                 pass
48.
49.         class MyCanvas(MyMplCanvas):
50.             """Classe para desenho da estrutra."""
51.
52.             def __init__(self, *args, **kwargs):
53.                 MyMplCanvas.__init__(self, *args, **kwargs)
54.                 self.diagramas = []
55.
```

```
56.             def offset(self, x, y):
57.                 return offset_copy(self.axes.transData, x=x, y=y, units='dots')
58.
59.             def desenhar_apoio1(self, x, y) :
60.
61.                 verts = [
62.                     (x, y), # left, bottom
63.                     (x-0.2, y-0.35), # left, top
64.                     (x+0.2, y-0.35), # right, top
65.                     (x, y), # ignored
66.                     (x-0.25, y-0.45),
67.                     (x+0.25, y-0.45)
68.                 ]
69.
70.                 codes = [
71.                     Path.MOVETO,
72.                     Path.LINETO,
73.                     Path.LINETO,
74.                     Path.CLOSEPOLY,
75.                     Path.MOVETO,
76.                     Path.LINETO
77.                 ]
78.
79.                 arr_patches = []
80.
81.                 path = Path(verts, codes)
82.
83.                 patch = patches.PathPatch(path)
84.
85.                 arr_patches.append(patch)
86.
87.                 circle1 = patches.Circle([x-0.14,y-0.40], 0.05)
88.
89.                 arr_patches.append(circle1)
90.
91.                 circle2 = patches.Circle([x+0.14,y-0.40], 0.05)
92.
93.                 arr_patches.append(circle2)
94.
95.                 collection = PatchCollection(arr_patches, facecolors = ((1,1,1,1),(1,
1,1,1),(1,1,1,1)), edgecolors = ((0,0,0,1),(0,0,0,1),(0,0,0,1)), linewidths = (0.5,0.5,0.
5))
96.
97.                 self.axes.add_collection(collection)
98.
99.             def desenhar_apoio2(self, x, y) :
100.
101.                 verts = [
102.                     (x, y), # left, bottom
103.                     (x-0.2, y-0.35), # left, top
104.                     (x+0.2, y-0.35), # right, top
105.                     (x, y), # ignored
106.                     (x-0.25, y-0.35),
107.                     (x+0.25, y-0.35),
108.                     (x-0.27, y-0.50),
109.                     (x-0.19, y-0.35),
110.                     (x-0.20, y-0.50),
111.                     (x-0.12, y-0.35),
112.                     (x-0.13, y-0.50),
113.                     (x-0.05, y-0.35),
114.                     (x-0.06, y-0.50),
115.                     (x+0.02, y-0.35),
116.                     (x+0.01, y-0.50),
117.                     (x+0.09, y-0.35),
118.                     (x+0.08, y-0.50),
119.                     (x+0.16, y-0.35),
```

```
120.                     (x+0.15, y-0.50),
121.                     (x+0.23, y-0.35)
122.                 ]
123.
124.                 codes = [
125.                     Path.MOVETO,
126.                     Path.LINETO,
127.                     Path.LINETO,
128.                     Path.CLOSEPOLY,
129.                     Path.MOVETO,
130.                     Path.LINETO,
131.                     Path.MOVETO,
132.                     Path.LINETO,
133.                     Path.MOVETO,
134.                     Path.LINETO,
135.                     Path.MOVETO,
136.                     Path.LINETO,
137.                     Path.MOVETO,
138.                     Path.LINETO,
139.                     Path.MOVETO,
140.                     Path.LINETO,
141.                     Path.MOVETO,
142.                     Path.LINETO,
143.                     Path.MOVETO,
144.                     Path.LINETO
145.                 ]
146.
147.                 path = Path(verts, codes)
148.
149.                 patch = patches.PathPatch(path, facecolor='white', lw=0.5)
150.                 self.axes.add_patch(patch)
151.
152.             def desenhar_apoio3(self, x, y) :
153.
154.                 verts = [
155.                     (x-0.25, y),
156.                     (x+0.25, y),
157.                     (x-0.27, y-0.15),
158.                     (x-0.19, y),
159.                     (x-0.20, y-0.15),
160.                     (x-0.12, y),
161.                     (x-0.13, y-0.15),
162.                     (x-0.05, y),
163.                     (x-0.06, y-0.15),
164.                     (x+0.02, y),
165.                     (x+0.01, y-0.15),
166.                     (x+0.09, y),
167.                     (x+0.08, y-0.15),
168.                     (x+0.16, y),
169.                     (x+0.15, y-0.15),
170.                     (x+0.23, y)
171.                 ]
172.
173.                 codes = [
174.                     Path.MOVETO,
175.                     Path.LINETO,
176.                     Path.MOVETO,
177.                     Path.LINETO,
178.                     Path.MOVETO,
179.                     Path.LINETO,
180.                     Path.MOVETO,
181.                     Path.LINETO,
182.                     Path.MOVETO,
183.                     Path.LINETO,
184.                     Path.MOVETO,
185.                     Path.LINETO,
```

```
186.                     Path.MOVETO,
187.                     Path.LINETO,
188.                     Path.MOVETO,
189.                     Path.LINETO
190.                 ]
191.
192.                 path = Path(verts, codes)
193.
194.                 patch = patches.PathPatch(path, facecolor='goldenrod', lw=0.5)
195.                 self.axes.add_patch(patch)
196.
197.             def desenhar_carga_vertical(self, x, y, f) :
198.
199.                 if f < 0:
200.                     a = +1.5
201.                 elif f > 0:
202.                     a = -1.5
203.
204.                 patch = patches.FancyArrowPatch(
205.                     (x, y+a),
206.                     (x, y),
207.                     arrowstyle='->',
208.                     mutation_scale=15, color="goldenrod", zorder = 101, alpha=0.8, lw
=1.3
209.                 )
210.
211.                 self.axes.add_patch(patch)
212.
213.                 text = self.axes.annotate("{0:.2f} kN".format(f), xy=(x, y+a/2), xyte
xt=(7, 0), textcoords='offset points', size=9, color="goldenrod", zorder=1, horizontalali
gnment="center", verticalalignment='center')
214.                 text.set_rotation(90)
215.                 self.draw()
216.
217.             def desenhar_carga_horizontal(self, x, y, f) :
218.
219.                 if f < 0:
220.                     a = +1.5
221.                 elif f > 0:
222.                     a = -1.5
223.
224.                 patch = patches.FancyArrowPatch(
225.                     (x+a, y),
226.                     (x, y),
227.                     arrowstyle='->',
228.                     mutation_scale=15, color="goldenrod", zorder = 101, alpha=0.8, lw
=1.3
229.                 )
230.
231.                 self.axes.add_patch(patch)
232.
233.                 text = self.axes.annotate("{0:.2f} kN".format(f), xy=(x+a/2, y), xyte
xt=(0, 7), textcoords='offset points', size=9, color="goldenrod", zorder=1, horizontalali
gnment="center", verticalalignment='center')
234.                 text.set_rotation(0)
235.                 self.draw()
236.
237.             def desenhar_carga_distribuida2(self, x_1, y_1, x_2, y_2, qx_1, qx_2, qy_
1, qy_2) :
238.
239.                 max = 10
240.
241.                 theta = math.atan2(y_2 - y_1, x_2 - x_1)
242.                 L = math.sqrt((x_2 - x_1)**2 + (y_2 - y_1)**2)
243.                 x = x_1
244.                 y = y_1
```

```
245.                 c = 0.5
246.                 l = 0
247.
248.                 N = math.floor(L/c)
249.                 c = L/N
250.
251.
252.
253.                 while l <= L + 0.001:
254.
255.                     q = qy_1 + (qy_2 - qy_1) * (l/L)
256.
257.                     #s = 1 + math.floor((q/max)*10)
258.                     patch = patches.FancyArrowPatch((x + (q/max)*math.cos(math.pi/2 +
 theta), y + (q/max)*math.sin(math.pi/2 + theta)),
259.                         (x, y),
260.                         arrowstyle='->',
261.                         mutation_scale=2, color = 'goldenrod', shrinkA = 0, shrinkB =
 0, alpha = 0.8
262.                     )
263.                     self.axes.add_patch(patch)
264.
265.                     x = x + c*math.cos(theta)
266.                     y = y + c*math.sin(theta)
267.                     l = math.sqrt((x - x_1)**2 + (y - y_1)**2)
268.
269.                 xof = 5 * math.cos(math.pi/2 + theta)
270.                 yof = 5 * math.sin(math.pi/2 + theta)
271.
272.                 xt_1 = x_1 + (qy_1/max) * math.cos(math.pi/2 + theta)
273.                 yt_1 = y_1 + (qy_1/max) * math.sin(math.pi/2 + theta)
274.                 xt_2 = x_2 + (qy_2/max) * math.cos(math.pi/2 + theta)
275.                 yt_2 = y_2 + (qy_2/max) * math.sin(math.pi/2 + theta)
276.
277.                 beta = math.atan2(yt_2 - yt_1, xt_2 - xt_1)
278.                 rotation = beta * 180/math.pi
279.
280.                 if qy_1 > 0.01:
281.                     text_1 = self.axes.annotate("{0:.2f} [kN/m]".format(qy_1), xy=(xt
_1, yt_1), xytext=(xof, yof), textcoords='offset points', size=9, color="goldenrod", zord
er=1, horizontalalignment="center", verticalalignment='center')
282.                     text_1.set_rotation(rotation)
283.
284.                 if qy_2 > 0.01:
285.                     text_2 = self.axes.annotate("{0:.2f} [kN/m]".format(qy_2), xy=(xt
_2, yt_2), xytext=(xof, yof), textcoords='offset points', size=9, color="goldenrod", zord
er=1, horizontalalignment="center", verticalalignment='center')
286.                     text_2.set_rotation(rotation)
287.
288.                 line = Line2D((x_1+(qy_1/max)*math.cos(math.pi/2 + theta),x_2+(qy_2/m
ax)*math.cos(math.pi/2 + theta)), (y_1+(qy_1/max)*math.sin(math.pi/2 + theta),y_2+(qy_2/m
ax)*math.sin(math.pi/2 + theta)), linewidth=1.0, color="goldenrod", alpha = 0.8)
289.                 self.axes.add_line(line)
290.                 self.draw()
291.
292.             def desenhar_carga_fletora(self, x, y, m) :
293.
294.                 trans = self.offset(10,0)
295.
296.                 radius = 0.75
297.                 angle = -30
298.                 theta2 = 240
299.                 color = 'goldenrod'
300.                 alpha = 0.8
301.
302.                 centX = x
```

```
303.                centY = y
304.
305.                arc = Arc([x,y],radius,radius,angle=angle,theta1=0,theta2=theta2,caps
tyle='round',linestyle='-',lw=1,color=color, alpha=alpha)
306.                self.axes.add_patch(arc)
307.
308.                startX = centX+(radius/2)*np.cos(rad(angle))
309.                startY = centY+(radius/2)*np.sin(rad(angle))
310.
311.                startXa = centX+(radius/2)*np.cos(rad(angle-0.000005))
312.                startYa = centY+(radius/2)*np.sin(rad(angle-0.000005))
313.
314.                endX = centX+(radius/2)*np.cos(rad(theta2+angle))
315.                endY = centY+(radius/2)*np.sin(rad(theta2+angle))
316.
317.                endXa = centX+(radius/2)*np.cos(rad(theta2+angle+0.000005))
318.                endYa = centY+(radius/2)*np.sin(rad(theta2+angle+0.000005))
319.
320.
321.                if m < 0:
322.                    xi = startX
323.                    xf = startXa
324.                    yi = startY
325.                    yf = startYa
326.                elif m > 0:
327.                    xi = endX
328.                    xf = endXa
329.                    yi = endY
330.                    yf = endYa
331.
332.                patch = patches.FancyArrowPatch((xi, yi), (xf, yf), arrowstyle='-
>',
333.                mutation_scale=15, color = 'goldenrod', shrinkA = 0, shrinkB = 0, alp
ha = 0.8)
334.
335.                self.axes.add_patch(patch)
336.                self.axes.text(x, y+0.35, "{0:.2f} kN.m".format(m), horizontalalignme
nt='center', verticalalignment='bottom', color='goldenrod', size = 9, transform=trans)
337.                self.draw()
338.
339.            def compute_initial_figure(self):
340.
341.                self.axes.set_xlabel('x')
342.                self.axes.set_ylabel('y')
343.
344.                spacing = 1.0
345.                self.axes.set_aspect('equal', 'datalim')
346.                minorLocator = MultipleLocator(spacing)
347.                self.axes.yaxis.set_minor_locator(minorLocator)
348.                self.axes.xaxis.set_minor_locator(minorLocator)
349.                self.axes.grid(which = 'minor')
350.
351.            def desenhar_no(self, r, x, y):
352.                trans = self.offset(0,0)
353.                self.axes.scatter(x, y, alpha=0.5, marker='o', zorder = 99, s=150, ed
gecolor='red', facecolor='white')
354.                self.axes.text(x, y, str(r), color='r', size=9, zorder = 100, horizon
talalignment='center', verticalalignment='center', transform=trans)
355.                self.draw()
356.
357.            def desenhar_carga(self, x, y, f_x, f_y, m_z):
358.                if f_x :
359.                    self.desenhar_carga_horizontal(x,y,f_x)
360.                if f_y :
361.                    self.desenhar_carga_vertical(x,y,f_y)
362.                if m_z :
```

```
363.                     self.desenhar_carga_fletora(x,y,m_z)
364.
365.             def desenhar_apoio(self, x, y, deslocamento_x, deslocamento_y, rotacao_z)
:
366.                 if (deslocamento_x == 0) and (deslocamento_y == 0) and (rotacao_z ==
0) :
367.                     self.desenhar_apoio3(x, y)
368.                 elif (deslocamento_x == 0) and (deslocamento_y == 0) and (rotacao_z =
= 1) :
369.                     self.desenhar_apoio2(x, y)
370.                 elif (deslocamento_x == 1) and (deslocamento_y == 0) and (rotacao_z =
= 1) :
371.                     self.desenhar_apoio1(x, y)
372.         #         self.draw()
373.
374.             def desenhar_barra(self, idc, barra):
375.
376.                 x_1 = barra.no_1.x
377.                 y_1 = barra.no_1.y
378.                 x_2 = barra.no_2.x
379.                 y_2 = barra.no_2.y
380.
381.                 rotula_1 = barra.condicao_rotulas[0]
382.                 rotula_2 = barra.condicao_rotulas[1]
383.
384.                 theta = math.atan2(y_2 - y_1, x_2 - x_1)
385.
386.                 r = 0.15
387.
388.                 if (rotula_1 == 0) :
389.                     self.axes.scatter(x_1 + r*math.cos(theta), y_1 + r*math.sin(theta
), alpha=0.5, marker='o', zorder = 90, s=50, edgecolor='black', facecolor='white')
390.
391.                 if (rotula_2 == 0) :
392.                     self.axes.scatter(x_2 - r*math.cos(theta), y_2 - r*math.sin(theta
), alpha=0.5, marker='o', zorder = 90, s=50, edgecolor='black', facecolor='white')
393.
394.                 xs = [x_1, x_2]
395.                 ys = [y_1, y_2]
396.                 line = Line2D(xs, ys, linewidth=1.0, color='blue')
397.                 self.axes.add_line(line)
398.
399.                 trans = self.offset(0,0)
400.
401.                 xm = (x_1 + x_2)/2
402.                 ym = (y_1 + y_2)/2
403.                 self.axes.scatter(xm, ym, alpha=0.5, marker='o', zorder = 99, s=150,
edgecolor='blue', facecolor='white')
404.                 self.axes.text(xm, ym, str(idc), color='b', size=9, zorder = 100, hor
izontalalignment='center', verticalalignment='center', transform=trans)
405.                 self.draw()
406.
407.             def desenhar_estrutura(self, estrutura):
408.                 for i, no in enumerate(estrutura.nos):
409.                     self.desenhar_no(i, no.x, no.y)
410.                     self.desenhar_apoio(no.x, no.y, no.condicao_apoio[0], no.condicao
_apoio[1], no.condicao_apoio[2])
411.                     self.desenhar_carga(no.x, no.y, no.carga_resultante.fx, no.carga_
resultante.fy, no.carga_resultante.mz)
412.
413.                 for i, barra in enumerate(estrutura.barras):
414.                     self.desenhar_barra(i, barra)
415.
416.                     for carga_distribuida in barra.cargas_distribuidas:
417.                         qx_1 = carga_distribuida.qx_1
418.                         qx_2 = carga_distribuida.qx_2
```

```
419.                         qy_1 = carga_distribuida.qy_1
420.                         qy_2 = carga_distribuida.qy_2
421.                         x_1 = barra.no_1.x
422.                         x_2 = barra.no_2.x
423.                         y_1 = barra.no_1.y
424.                         y_2 = barra.no_2.y
425.                         self.desenhar_carga_distribuida2(x_1, y_1, x_2, y_2, qx_1, qx
_2, qy_1, qy_2)
426.
427.             def desenhar_no_deformada(self, r, x, y):
428.                 trans = self.offset(0,0)
429.                 self.axes.scatter(x, y, alpha=0.5, marker='o', zorder = 99, s=50, edg
ecolor='gray', facecolor='white')
430.                 self.axes.text(x, y, str(r), color='gray', size=7, zorder = 100, hori
zontalalignment='center', verticalalignment='center', transform=trans)
431.                 self.draw()
432.
433.             def reiniciar_diagramas(self):
434.                 for d in self.diagramas:
435.                    d.remove()
436.                 self.diagramas = []
437.
438.             def desenhar_deformacoes(self, estrutura, estrutura_discretizada):
439.                 self.reiniciar_diagramas()
440.
441.                 mat_abs_1 = np.absolute(estrutura_discretizada.no_disp[:,0])
442.                 mat_abs_2 = np.absolute(estrutura_discretizada.no_disp[:,1])
443.                 max = np.amax(mat_abs_1.tolist() + mat_abs_2.tolist())
444.
445.                 for idc_barra, barra in enumerate(estrutura.barras):
446.                     indices = estrutura_discretizada.subnos_barras_discretizados[idc_
barra]
447.                     coord_x = []
448.                     coord_y = []
449.
450.                     for k, i in enumerate(indices):
451.                         no_discretizada = estrutura_discretizada.nos[i-1]
452.                         x = no_discretizada.x
453.                         y = no_discretizada.y
454.                         d_x = 1.0 * estrutura_discretizada.no_disp[i-1][0]/max
455.                         d_y = 1.0 * estrutura_discretizada.no_disp[i-1][1]/max
456.
457.                         deslocamento = math.sqrt(d_x ** 2 + d_y ** 2)
458.                         coord_x.append(x + d_x)
459.                         coord_y.append(y + d_y)
460.
461.                         if k == 0 or k == 2 or k == 4:
462.                             trans = self.offset(0,0)
463.                             d = self.axes.text(x + d_x, y + d_y, "{0:.2f}".format(des
locamento), color='gray', size=7, zorder = 100, horizontalalignment='center', verticalali
gnment='center', transform=trans)
464.                             self.diagramas.append(d)
465.
466.                     tck, u = interpolate.splprep([coord_x,coord_y], u=None, s=0.0)
467.                     u_new = np.linspace(u.min(), u.max(), 1000)
468.                     x_new, y_new = interpolate.splev(u_new, tck, der=0)
469.
470.                     d, = self.axes.plot(x_new, y_new, '-', color='silver')
471.                     self.diagramas.append(d)
472.
473.                 self.draw()
474.
475.             def desenhar_dmf(self, estrutura, estrutura_discretizada):
476.                 self.reiniciar_diagramas()
477.
478.                 mat_abs_1 = np.absolute(estrutura_discretizada.forca_l[:,2])
```

```
479.                mat_abs_2 = np.absolute(estrutura_discretizada.forca_l[:,5])
480.                max = np.amax(mat_abs_1.tolist() + mat_abs_2.tolist())
481.
482.                if max < 0.01 : max = 0.01
483.
484.                for idc_barra, barra in enumerate(estrutura.barras):
485.                    indices = estrutura_discretizada.subbarras_barras_discretizados[i
dc_barra]
486.                    coord_x = []
487.                    coord_y = []
488.
489.                    x1 = barra.no_1.x
490.                    x2 = barra.no_2.x
491.                    y1 = barra.no_1.y
492.                    y2 = barra.no_2.y
493.
494.                    theta = math.atan2(y2 - y1, x2 - x1)
495.
496.                    L = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
497.
498.                    rotation = theta * 180/math.pi
499.
500.                    for k, i in enumerate(indices):
501.                        subbarra = estrutura_discretizada.barras[i-1]
502.                        m = estrutura_discretizada.forca_l[i-1,2]
503.                        x_t = subbarra.no_1.x + (m*1.5/max) * math.cos(math.pi/2 + th
eta)
504.                        y_t = subbarra.no_1.y + (m*1.5/max) * math.sin(math.pi/2 + th
eta)
505.                        print((m*1.5/max) * math.cos(math.pi/2 + theta))
506.                        coord_x.append(x_t)
507.                        coord_y.append(y_t)
508.
509.                        if k == 3:
510.                            m = (-1)*estrutura_discretizada.forca_l[i-1,5]
511.                            x_t = subbarra.no_2.x + (m*1.5/max) * math.cos(math.pi/2
+ theta)
512.                            y_t = subbarra.no_2.y + (m*1.5/max) * math.sin(math.pi/2
+ theta)
513.                            coord_x.append(x_t)
514.                            coord_y.append(y_t)
515.
516.                        if (k == 0 or k == 2 or k == 3) and m > 0.01:
517.                            trans = self.offset(0,0)
518.                            d = self.axes.text(x_t, y_t, "{0:.2f}".format(m), color='
green', size=7, zorder = 100, horizontalalignment='center', verticalalignment='center', t
ransform=trans)
519.                            self.diagramas.append(d)
520.
521.                    if max > 0.01 :
522.                        tck, u = interpolate.splprep([coord_x,coord_y], u=None, s=0.0
)
523.                        u_new = np.linspace(u.min(), u.max(), 100)
524.                        x_new, y_new = interpolate.splev(u_new, tck, der=0)
525.
526.                        d, = self.axes.plot(x_new, y_new, '-
', color='green', lw=1, alpha=0.1)
527.                        self.diagramas.append(d)
528.
529.                        x_k = [x1] + x_new.tolist() + [x2]
530.                        y_k = [y1] + y_new.tolist() + [y2]
531.
532.                        d, = self.axes.fill(x_k, y_k, 'g', alpha=0.05, edgecolor='g')

533.                        self.diagramas.append(d)
534.
```

```
535.                 self.draw()
536.
537.             def desenhar_dec(self, estrutura, estrutura_discretizada):
538.                 self.reiniciar_diagramas()
539.
540.                 mat_abs_1 = np.absolute(estrutura_discretizada.forca_l[:,1])
541.                 mat_abs_2 = np.absolute(estrutura_discretizada.forca_l[:,4])
542.                 max = np.amax(mat_abs_1.tolist() + mat_abs_2.tolist())
543.
544.                 if max < 0.01 : max = 0.01
545.
546.                 for idc_barra, barra in enumerate(estrutura.barras):
547.                     indices = estrutura_discretizada.subbarras_barras_discretizados[i
dc_barra]
548.                     coord_x = []
549.                     coord_y = []
550.
551.                     x1 = barra.no_1.x
552.                     x2 = barra.no_2.x
553.                     y1 = barra.no_1.y
554.                     y2 = barra.no_2.y
555.
556.                     theta = math.atan2(y2 - y1, x2 - x1)
557.
558.                     L = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
559.
560.                     rotation = theta * 180/math.pi
561.
562.                     for k, i in enumerate(indices):
563.                         subbarra = estrutura_discretizada.barras[i-1]
564.                         m = estrutura_discretizada.forca_l[i-1,1]
565.                         x_t = subbarra.no_1.x + (m*1.5/max) * math.cos(math.pi/2 + th
eta)
566.                         y_t = subbarra.no_1.y + (m*1.5/max) * math.sin(math.pi/2 + th
eta)
567.                         coord_x.append(x_t)
568.                         coord_y.append(y_t)
569.
570.                         if k == 3:
571.                             m = (-1)*estrutura_discretizada.forca_l[i-1,4]
572.                             x_t = subbarra.no_2.x + (m*1.5/max) * math.cos(math.pi/2
+ theta)
573.                             y_t = subbarra.no_2.y + (m*1.5/max) * math.sin(math.pi/2
+ theta)
574.                             coord_x.append(x_t)
575.                             coord_y.append(y_t)
576.
577.                         if (k == 0 or k == 2 or k == 3) and m > 0.01:
578.                             trans = self.offset(0,0)
579.                             d = self.axes.text(x_t, y_t, "{0:.2f}".format(m), color='
orange', size=7, zorder = 100, horizontalalignment='center', verticalalignment='center',
transform=trans)
580.                             self.diagramas.append(d)
581.
582.                     if max > 0.01 :
583.                         tck, u = interpolate.splprep([coord_x,coord_y], u=None, s=0.0
)
584.                         u_new = np.linspace(u.min(), u.max(), 100)
585.                         x_new, y_new = interpolate.splev(u_new, tck, der=0)
586.
587.                         d, = self.axes.plot(x_new, y_new, '-
', color='orange', lw=1, alpha=0.1)
588.                         self.diagramas.append(d)
589.                         x_k = [x1] + x_new.tolist() + [x2]
590.                         y_k = [y1] + y_new.tolist() + [y2]
591.
```

```
592.                         d, = self.axes.fill(x_k, y_k, color='orange', alpha=0.05, edg
ecolor='orange')
593.                         self.diagramas.append(d)
594.
595.                 self.draw()
596.
597.             def desenhar_den(self, estrutura, estrutura_discretizada):
598.                 self.reiniciar_diagramas()
599.
600.                 mat_abs_1 = np.absolute(estrutura_discretizada.forca_l[:,0])
601.                 mat_abs_2 = np.absolute(estrutura_discretizada.forca_l[:,3])
602.                 max = np.amax(mat_abs_1.tolist() + mat_abs_2.tolist())
603.
604.                 if max < 0.01 : max = 0.01
605.
606.                 for idc_barra, barra in enumerate(estrutura.barras):
607.                     indices = estrutura_discretizada.subbarras_barras_discretizados[i
dc_barra]
608.                     coord_x = []
609.                     coord_y = []
610.
611.                     x1 = barra.no_1.x
612.                     x2 = barra.no_2.x
613.                     y1 = barra.no_1.y
614.                     y2 = barra.no_2.y
615.
616.                     theta = math.atan2(y2 - y1, x2 - x1)
617.
618.                     L = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
619.
620.                     rotation = theta * 180/math.pi
621.
622.                     for k, i in enumerate(indices):
623.                         subbarra = estrutura_discretizada.barras[i-1]
624.                         m = estrutura_discretizada.forca_l[i-1,0]
625.                         x_t = subbarra.no_1.x + (m*1.5/max) * math.cos(math.pi/2 + th
eta)
626.                         y_t = subbarra.no_1.y + (m*1.5/max) * math.sin(math.pi/2 + th
eta)
627.                         coord_x.append(x_t)
628.                         coord_y.append(y_t)
629.
630.                         if k == 3:
631.                             m = (-1)*estrutura_discretizada.forca_l[i-1,3]
632.                             x_t = subbarra.no_2.x + (m*1.5/max) * math.cos(math.pi/2
+ theta)
633.                             y_t = subbarra.no_2.y + (m*1.5/max) * math.sin(math.pi/2
+ theta)
634.                             coord_x.append(x_t)
635.                             coord_y.append(y_t)
636.
637.                         if (k == 0 or k == 2 or k == 3) and m > 0.01:
638.                             trans = self.offset(0,0)
639.                             d = self.axes.text(x_t, y_t, "{0:.2f}".format(m), color='
red', size=7, zorder = 100, horizontalalignment='center', verticalalignment='center', tra
nsform=trans)
640.                             self.diagramas.append(d)
641.
642.                     if max > 0.01 :
643.                         tck, u = interpolate.splprep([coord_x,coord_y], u=None, s=0.0
)
644.                         u_new = np.linspace(u.min(), u.max(), 100)
645.                         x_new, y_new = interpolate.splev(u_new, tck, der=0)
646.
647.                         d, = self.axes.plot(x_new, y_new, '-
', color='red', lw=1, alpha=0.1)
```

```
                        self.diagramas.append(d)

                        x_k = [x1] + x_new.tolist() + [x2]
                        y_k = [y1] + y_new.tolist() + [y2]

                        d, = self.axes.fill(x_k, y_k, 'red', alpha=0.05, edgecolor='r
ed')
                        self.diagramas.append(d)

                self.draw()
```

B.2 Classe MainWindow (Janela Principal)

```
        from __future__ import unicode_literals

        import sys
        import os
        import random
        import matplotlib
        import math

        matplotlib.use('Qt5Agg')
        from PyQt5 import QtCore, QtGui, QtWidgets
        from PyQt5.QtCore import QDate, QFile, Qt, QTextStream
        from PyQt5.QtGui import (QFont, QIcon, QKeySequence, QTextCharFormat, QTextCu
rsor, QTextTableFormat)
        from PyQt5.QtPrintSupport import QPrintDialog, QPrinter
        from PyQt5.QtWidgets import (QWidget, QAction, QApplication, QDialog, QDockWi
dget, QTabWidget,
                QFileDialog, QListWidget, QMainWindow, QMessageBox, QTextEdit)

        from numpy import arange, sin, pi
        from numpy.random import rand
        from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as Naviga
tionToolbar
        from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCan
vas
        from matplotlib.figure import Figure
        from matplotlib.ticker import MultipleLocator
        from matplotlib.lines import Line2D
        from matplotlib.path import Path
        from matplotlib.collections import PatchCollection
        import matplotlib.patches as patches
        import matplotlib.pyplot as plt

        from componentes.estrutura.estrutura import Estrutura
        from .canvas import MyCanvas
        from .painel_dados_gerais import PainelDadosGerais
        from .painel_estrutura import PainelEstrutura
        from .painel_cargas import PainelCargas
        from .painel_restricoes import PainelRestricoes

        from reportlab.pdfgen import canvas
        from reportlab.pdfbase import pdfmetrics
        from reportlab.pdfbase.ttfonts import TTFont

        progname = os.path.basename(sys.argv[0])
        progversion = "0.1"

        class MainWindow(QMainWindow):
            def __init__(self):
                super(MainWindow, self).__init__()

```

```
47.                 self.estrutura = Estrutura()
48.                 self.main_widget = QtWidgets.QWidget(self)
49.                 self.plot_canvas = MyCanvas(self, width=5, height=4, dpi=100)
50.                 self.navi_toolbar = NavigationToolbar(self.plot_canvas, self)
51.
52.                 l = QtWidgets.QVBoxLayout(self.main_widget)
53.                 l.addWidget(self.plot_canvas)
54.                 l.addWidget(self.navi_toolbar)
55.
56.                 self.main_widget.setFocus()
57.                 self.setCentralWidget(self.main_widget)
58.
59.                 self.createActions()
60.                 self.createMenus()
61.                 self.createDockMenu()
62.
63.                 self.setWindowTitle("DM")
64.
65.             def closeEvent(self,event):
66.                 result = QtWidgets.QMessageBox.question(self,
67.                               "Confirmar saída...",
68.                               "Você tem certeza que quer sair?",
69.                               QtWidgets.QMessageBox.Yes| QtWidgets.QMessageBox.No)
70.                 event.ignore()
71.
72.                 if result == QtWidgets.QMessageBox.Yes:
73.                     event.accept()
74.
75.             def novo(self):
76.                 self.plot_canvas.limpar()
77.                 self.estrutura = Estrutura()
78.
79.             def open(self):
80.                 filename, _ = QFileDialog.getOpenFileName(self,
81.                         "Escolha o nome do arquivo", '.', "EST (*.est)")
82.                 if not filename:
83.                     return
84.
85.                 with open(filename) as f:
86.                     for line in f:
87.
88.                         line = line.replace("\n","")
89.                         line = line.replace(" ","")
90.
91.                         if "#secoes" in line:
92.                             marcador = "#secoes"
93.                             continue
94.                         elif "#materiais" in line:
95.                             marcador = "#materiais"
96.                             continue
97.                         elif "#nos" in line:
98.                             marcador = "#nos"
99.                             continue
100.                        elif "#barras" in line:
101.                            marcador = "#barras"
102.                            continue
103.                        elif "#condicoes_contorno" in line:
104.                            marcador = "#condicoes_contorno"
105.                            continue
106.                        elif "#rotulas" in line:
107.                            marcador = "#rotulas"
108.                            continue
109.                        elif "#cargas_nos" in line:
110.                            marcador = "#cargas_nos"
111.                            continue
112.                        elif "#cargas_distribuidas" in line:
```

```
113.                             marcador = "#cargas_distribuidas"
114.                             continue
115.
116.                         if (not marcador) or (not line):
117.                             continue
118.
119.                         if "#secoes" in marcador:
120.                             params = line.split(";")
121.                             A = float(params[0])
122.                             I = float(params[1])
123.                             self.estrutura.adicionar_secao(A,I)
124.                         elif "#materiais" in marcador:
125.                             params = line.split(";")
126.                             E = float(params[0])
127.                             self.estrutura.adicionar_material(E)
128.                         elif "#nos" in marcador:
129.                             params = line.split(";")
130.                             x = float(params[0])
131.                             y = float(params[1])
132.                             self.estrutura.adicionar_no(x,y)
133.                         elif "#barras" in marcador:
134.                             params = line.split(";")
135.                             idc_no_inicial = int(params[0])
136.                             idc_no_final = int(params[1])
137.                             idc_material = int(params[2])
138.                             idc_secao = int(params[3])
139.                             self.estrutura.adicionar_barra(idc_no_inicial, idc_no_fin
al, idc_material, idc_secao)
140.                         elif "#condicoes_contorno" in marcador:
141.                             params = line.split(";")
142.                             idc_no = int(params[0])
143.                             deslocamento_x = int(params[1])
144.                             deslocamento_y = int(params[2])
145.                             rotacao_z = int(params[3])
146.                             self.estrutura.especificar_condicao_contorno(idc_no, desl
ocamento_x, deslocamento_y, rotacao_z)
147.                         elif "#rotulas" in marcador:
148.                             params = line.split(";")
149.                             idc_barra = int(params[0])
150.                             rotula_1 = int(params[1])
151.                             rotula_2 = int(params[2])
152.                             self.estrutura.especificar_condicao_rotulas(idc_barra, ro
tula_1, rotula_2)
153.                         elif "#cargas_nos" in marcador:
154.                             params = line.split(";")
155.                             idc_no = int(params[0])
156.                             f_x = int(params[1])
157.                             f_y = int(params[2])
158.                             m_z = int(params[3])
159.                             self.estrutura.adicionar_carga_no(idc_no, f_x, f_y, m_z)

160.                         elif "#cargas_distribuidas" in marcador:
161.                             params = line.split(";")
162.                             idc_barra = int(params[0])
163.                             qx_1 = int(params[1])
164.                             qx_2 = int(params[2])
165.                             qy_1 = int(params[3])
166.                             qy_2 = int(params[4])
167.                             self.estrutura.adicionar_carga_distribuida(idc_barra, qx_
1, qx_2, qy_1, qy_2)
168.
169.                 self.plot_canvas.desenhar_estrutura(self.estrutura)
170.                 self.statusBar().showMessage("Estrutura carregada '%s'" % filename, 2
000)
171.
172.             def save(self):
```

```
173.                 filename, _ = QFileDialog.getSaveFileName(self,
174.                         "Escolha o nome do arquivo", '.', "EST (*.est)")
175.                 if not filename:
176.                     return
177.
178.                 file = QFile(filename)
179.                 if not file.open(QFile.WriteOnly | QFile.Text):
180.                     QMessageBox.warning(self, "Dock Widgets", "Cannot write file %s:\
n%s." % (filename, file.errorString()))
181.                     return
182.
183.                 modelo = self.estrutura.gerar_modelo()
184.
185.                 out = QTextStream(file)
186.                 QApplication.setOverrideCursor(Qt.WaitCursor)
187.                 out << modelo
188.                 QApplication.restoreOverrideCursor()
189.
190.                 self.statusBar().showMessage("Salvo '%s'" % filename, 2000)
191.
192.             def about(self):
193.                 QMessageBox.about(self, "PyEstrutura2D",
194.                         "Trabalho de conclusão de curso apresentado ao "
195.                         "Departamento de Engenharia das Construções e Estruturas "
196.                         "da Universidade Estadual do Maranhão, como requisito para a
"
197.                         "obtenção do grau de Bacharel em Engenharia Civil. "
198.                         "O presente sistema é capaz de analisar estruturas reticulada
s "
199.                         "planas, apresentando os diagramas de deformações, momentos "

200.                         "fletores, esforços cortantes e normais. ")
201.
202.             def createActions(self):
203.                 self.newLetterAct = QAction(QIcon(':/images/new.png'), "&Novo",
204.                         self, shortcut=QKeySequence.New,
205.                         statusTip="Criar nova estrutura", triggered=self.novo)
206.
207.                 self.openAct = QAction(QIcon(':/images/open.png'), "&Abrir", self,
208.                         shortcut=QKeySequence.Open,
209.                         statusTip="Carregar estrutura", triggered=self.open)
210.
211.                 self.saveAct = QAction(QIcon(':/images/save.png'), "&Salvar", self,
212.                         shortcut=QKeySequence.Save,
213.                         statusTip="Salvar a estrutura atual", triggered=self.save)
214.
215.                 self.reportAct = QAction(QIcon(':/images/print.png'), "&Imprimir rela
tório", self,
216.                         shortcut="Ctrl+R",
217.                         statusTip="Imprime o relatório de análise da estrutura atual.
",
218.                         triggered=self.imprimirRelatorio)
219.
220.                 self.quitAct = QAction("&Sair", self, shortcut="Ctrl+Q",
221.                         statusTip="Sair da aplicativo", triggered=self.close)
222.
223.                 self.aboutAct = QAction("&Sobre", self,
224.                         statusTip="Sobre o aplicativo",
225.                         triggered=self.about)
226.
227.                 self.defAct = QAction("&Diagrama de deformação", self,
228.                         statusTip="Exibir deformações",
229.                         triggered=self.exibirDeformacoes)
230.                 self.dmfAct = QAction("&Diagrama de momentos fletores", self,
231.                         statusTip="Exibir momentos fletores",
232.                         triggered=self.exibirDMF)
```

```
233.                 self.decAct = QAction("&Diagrama de esforços cortantes", self,
234.                         statusTip="Exibir diagrama de esforços cortantes",
235.                         triggered=self.exibirDEC)
236.                 self.denAct = QAction("Diagrama de esforços normais", self,
237.                         statusTip="Exibir diagrama de esforços normais",
238.                         triggered=self.exibirDEN)
239. 
240.             def imprimirRelatorio(self):
241. 
242.                 self.estrutura.calcular()
243. 
244.                 pdfmetrics.registerFont(TTFont('Camingo Code Regular', 'fontes/Caming
oCode-Regular.ttf'))
245. 
246.                 arquivo_pdf, _ = QFileDialog.getSaveFileName(self, "Escolha o nome do
 arquivo", '.', "PDF (*.pdf)")
247. 
248.                 if not arquivo_pdf:
249.                     return
250. 
251.                 file = QFile(arquivo_pdf)
252.                 if not file.open(QFile.WriteOnly | QFile.Text):
253.                     QMessageBox.warning(self, "Dock Widgets", "Cannot write file %s:\
n%s." % (arquivo_pdf, file.errorString()))
254.                     return
255. 
256.                 arquivo_txt, extensao = os.path.splitext(arquivo_pdf)
257.                 arquivo_txt += ".txt"
258. 
259.                 self.estrutura.exportar_resultados(arquivo_txt)
260.                 c = canvas.Canvas(arquivo_pdf)
261. 
262.                 ptr = open(arquivo_txt, "r")
263.                 linhas = ptr.readlines()
264.                 ptr.close()
265. 
266.                 i = 750
267.                 numeroLinha = 0
268. 
269.                 while numeroLinha < len(linhas):
270. 
271.                     c.setFont('Camingo Code Regular', 12)
272. 
273.                     if numeroLinha - len(linhas) < 60:
274.                         i=750
275.                         for linha in linhas[numeroLinha:numeroLinha+60]:
276.                             c.drawString(15, i, linha.strip())
277.                             numeroLinha += 1
278.                             i -= 12
279.                         c.showPage()
280.                     else:
281.                         i = 750
282.                         for linha in linhas[numeroLinha:]:
283.                             c.drawString(15, i, linha.strip())
284.                             numeroLinha += 1
285.                             i -= 12
286.                         c.showPage()
287.                 c.save()
288. 
289.             def exibirDeformacoes(self):
290.                 disc = self.estrutura.discretizar()
291.                 disc.calcular()
292.                 self.plot_canvas.desenhar_deformacoes(self.estrutura, disc)
293. 
294.             def exibirDMF(self):
295.                 disc = self.estrutura.discretizar()
```

```
296.                 disc.calcular()
297.                 self.plot_canvas.desenhar_dmf(self.estrutura, disc)
298.
299.             def exibirDEC(self):
300.                 disc = self.estrutura.discretizar()
301.                 disc.calcular()
302.                 self.plot_canvas.desenhar_dec(self.estrutura, disc)
303.
304.             def exibirDEN(self):
305.                 disc = self.estrutura.discretizar()
306.                 disc.calcular()
307.                 self.plot_canvas.desenhar_den(self.estrutura, disc)
308.
309.             def createMenus(self):
310.                 self.fileMenu = self.menuBar().addMenu("&Arquivo")
311.                 self.fileMenu.addAction(self.newLetterAct)
312.                 self.fileMenu.addAction(self.openAct)
313.                 self.fileMenu.addAction(self.saveAct)
314.                 self.fileMenu.addSeparator()
315.                 self.fileMenu.addAction(self.quitAct)
316.
317.                 self.execMenu = self.menuBar().addMenu("&Executar")
318.                 self.execMenu.addAction(self.reportAct)
319.
320.                 self.viewMenu = self.menuBar().addMenu("&Visualizar")
321.                 self.viewMenu.addAction(self.defAct)
322.                 self.viewMenu.addAction(self.dmfAct)
323.                 self.viewMenu.addAction(self.decAct)
324.                 self.viewMenu.addAction(self.denAct)
325.
326.                 self.menuBar().addSeparator()
327.
328.                 self.helpMenu = self.menuBar().addMenu("&Ajuda")
329.                 self.helpMenu.addAction(self.aboutAct)
330.
331.             def createDockMenu(self):
332.
333.                 self.tabWidget = QTabWidget(self)
334.
335.                 widget = QWidget()
336.                 layout = QtWidgets.QVBoxLayout()
337.                 self.mainwindow1 = PainelDadosGerais(self)
338.                 layout.addWidget(self.mainwindow1)
339.                 widget.setLayout(layout)
340.                 self.tabWidget.addTab(widget,"Dados gerais")
341.
342.                 widget = QWidget()
343.                 layout = QtWidgets.QVBoxLayout()
344.                 self.mainwindow2 = PainelEstrutura(self)
345.                 layout.addWidget(self.mainwindow2)
346.                 widget.setLayout(layout)
347.                 self.tabWidget.addTab(widget,"Estrutura")
348.
349.                 widget = QWidget()
350.                 layout = QtWidgets.QVBoxLayout()
351.                 self.mainwindow3 = PainelCarga(self)
352.                 layout.addWidget(self.mainwindow3)
353.                 widget.setLayout(layout)
354.                 self.tabWidget.addTab(widget,"Cargas")
355.
356.                 widget = QWidget()
357.                 layout = QtWidgets.QVBoxLayout()
358.                 self.mainwindow4 = PainelRestricoes(self)
359.                 layout.addWidget(self.mainwindow4)
360.                 widget.setLayout(layout)
361.                 self.tabWidget.addTab(widget,"Restrições")
```

```
362.
363.                 dock2 = QDockWidget("Propriedades", self)
364.                 dock2.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)

365.                 dock2.setWidget(self.tabWidget)
366.                 dock2.setFeatures(QDockWidget.AllDockWidgetFeatures)
367.                 self.addDockWidget(Qt.RightDockWidgetArea, dock2)
368.
369.                 self.tabWidget.currentChanged.connect(self.tabSelected)
370.
371.             def tabSelected(self, selected_index):
372.                 if selected_index == 0:
373.                     self.mainwindow2.atualizar()
374.                 elif selected_index == 1:
375.                     self.mainwindow2.atualizar()
376.                 elif selected_index == 2:
377.                     self.mainwindow3.atualizar()
378.                 elif selected_index == 3:
379.                     self.mainwindow4.atualizar()
```

B.3 Classe Painel Cargas

```
1.          from PyQt5 import QtCore, QtGui, QtWidgets
2.          from PyQt5.QtCore import QDate, QFile, Qt, QTextStream
3.          from PyQt5.QtGui import (QFont, QIcon, QKeySequence, QTextCharFormat,
4.                  QTextCursor, QTextTableFormat)
5.          from PyQt5.QtPrintSupport import QPrintDialog, QPrinter
6.          from PyQt5.QtWidgets import (QWidget, QAction, QApplication, QDialog, QDockWi
dget, QTabWidget,
7.                  QFileDialog, QListWidget, QMainWindow, QMessageBox, QTextEdit)
8.
9.          class PainelCargas(QMainWindow):
10.             def __init__(self, main_window):
11.                 super(PainelCargas, self).__init__()
12.
13.                 self.main_window = main_window
14.                 self.estrutura = self.main_window.estrutura
15.                 self.plot_canvas = self.main_window.plot_canvas
16.
17.                 self.main_widget = QtWidgets.QWidget(self)
18.
19.                 self.createDockCargasConcentradas()
20.                 self.createDockCargasDistribuidas()
21.
22.             def createDockCargasConcentradas(self):
23.
24.                 s = QWidget()
25.
26.                 self.vbox_no = QtWidgets.QVBoxLayout()
27.                 self.rotulo_no = QtWidgets.QLabel("Nó:")
28.                 self.combo_no = QtWidgets.QComboBox()
29.                 self.combo_no.setMinimumHeight(21)
30.                 self.combo_no.setMaximumHeight(21)
31.                 self.combo_no.setMinimumWidth(90)
32.                 self.combo_no.setMaximumWidth(90)
33.                 self.vbox_no.addWidget(self.rotulo_no)
34.                 self.vbox_no.addWidget(self.combo_no)
35.
36.                 self.vbox_fx = QtWidgets.QVBoxLayout()
37.                 self.rotulo_fx = QtWidgets.QLabel("Fx (kN):")
38.                 self.texto_fx = QtWidgets.QLineEdit()
39.                 self.vbox_fx.addWidget(self.rotulo_fx)
40.                 self.vbox_fx.addWidget(self.texto_fx)
```

```
41.
42.                 self.vbox_fy = QtWidgets.QVBoxLayout()
43.                 self.rotulo_fy = QtWidgets.QLabel("Fy (kN):")
44.                 self.texto_fy = QtWidgets.QLineEdit()
45.                 self.vbox_fy.addWidget(self.rotulo_fy)
46.                 self.vbox_fy.addWidget(self.texto_fy)
47.
48.                 self.vbox_mz = QtWidgets.QVBoxLayout()
49.                 self.rotulo_mz = QtWidgets.QLabel("Mz (kN):")
50.                 self.texto_mz = QtWidgets.QLineEdit()
51.                 self.vbox_mz.addWidget(self.rotulo_mz)
52.                 self.vbox_mz.addWidget(self.texto_mz)
53.
54.                 self.botao_adicionar_carga_concentrada = QtWidgets.QPushButton("Adici
onar")
55.                 self.botao_adicionar_carga_concentrada.setMinimumHeight(21)
56.                 self.botao_adicionar_carga_concentrada.setMaximumHeight(21)
57.                 self.botao_adicionar_carga_concentrada.setMinimumWidth(70)
58.                 self.botao_adicionar_carga_concentrada.setMaximumWidth(70)
59.                 self.botao_adicionar_carga_concentrada.clicked.connect(self.adicionar
_carga_concentrada)
60.
61.                 self.tree_cargas_concentradas = QtWidgets.QTreeWidget()
62.                 self.headers = ["Nó", "Fx", "Fy", "Mz"]
63.                 self.tree_cargas_concentradas.setColumnCount(len(self.headers))
64.                 self.tree_cargas_concentradas.setHeaderLabels(self.headers)
65.                 self.tree_cargas_concentradas.setSelectionMode(QtWidgets.QAbstractIte
mView.ExtendedSelection)
66.                 self.tree_cargas_concentradas.setIndentation(0)
67.                 self.tree_cargas_concentradas.header().resizeSection(0, 60)
68.                 self.tree_cargas_concentradas.header().resizeSection(1, 50)
69.                 self.tree_cargas_concentradas.header().resizeSection(2, 50)
70.                 self.tree_cargas_concentradas.header().resizeSection(3, 50)
71.
72.                 self.vbox_lista_cargas_concentradas = QtWidgets.QVBoxLayout()
73.                 self.rotulo_lista_cargas_concentradas = QtWidgets.QLabel("Lista de ca
rgas concentradas:")
74.                 self.vbox_lista_cargas_concentradas.addWidget(self.rotulo_lista_carga
s_concentradas)
75.                 self.vbox_lista_cargas_concentradas.addWidget(self.tree_cargas_concen
tradas)
76.
77.                 layout = QtWidgets.QGridLayout()
78.
79.                 layout.setColumnStretch(0,5)
80.                 layout.setColumnStretch(1,5)
81.                 layout.setColumnStretch(2,5)
82.
83.                 layout.setRowStretch(0,1)
84.                 layout.setRowStretch(1,1)
85.                 layout.setRowStretch(2,1)
86.                 layout.setRowStretch(3,1)
87.
88.                 layout.addLayout(self.vbox_no,0,0,1,2)
89.                 layout.addLayout(self.vbox_fx,1,0)
90.                 layout.addLayout(self.vbox_fy,1,1)
91.                 layout.addLayout(self.vbox_mz,1,2)
92.                 layout.addWidget(self.botao_adicionar_carga_concentrada,2,2,1,1,QtCor
e.Qt.AlignRight)
93.                 layout.addLayout(self.vbox_lista_cargas_concentradas,3,0,1,3)
94.
95.                 s.setLayout( layout )
96.
97.                 dock = QDockWidget("Cargas Concentradas", self)
98.                 dock.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)
```

```
99.                 dock.setWidget(s)
100.                dock.setMinimumHeight(250)
101.                dock.setMaximumHeight(250)
102.                dock.setMinimumWidth(300)
103.                dock.setMaximumWidth(300)
104.                dock.setFloating(False)
105.                dock.setFeatures(QDockWidget.NoDockWidgetFeatures)
106.                self.addDockWidget(Qt.RightDockWidgetArea, dock)
107.
108.            def createDockCargasDistribuidas(self):
109.
110.                s = QWidget()
111.
112.                self.vbox_barra = QtWidgets.QVBoxLayout()
113.                self.rotulo_barra = QtWidgets.QLabel("Barra:")
114.                self.combo_barra = QtWidgets.QComboBox()
115.                self.combo_barra.setMinimumHeight(21)
116.                self.combo_barra.setMaximumHeight(21)
117.                self.combo_barra.setMinimumWidth(90)
118.                self.combo_barra.setMaximumWidth(90)
119.                self.vbox_barra.addWidget(self.rotulo_barra)
120.                self.vbox_barra.addWidget(self.combo_barra)
121.
122.                self.vbox_qx1 = QtWidgets.QVBoxLayout()
123.                self.rotulo_qx1 = QtWidgets.QLabel("Qx1 (kN):")
124.                self.texto_qx1 = QtWidgets.QLineEdit()
125.                self.vbox_qx1.addWidget(self.rotulo_qx1)
126.                self.vbox_qx1.addWidget(self.texto_qx1)
127.
128.                self.vbox_qy1 = QtWidgets.QVBoxLayout()
129.                self.rotulo_qy1 = QtWidgets.QLabel("Qy1 (kN):")
130.                self.texto_qy1 = QtWidgets.QLineEdit()
131.                self.vbox_qy1.addWidget(self.rotulo_qy1)
132.                self.vbox_qy1.addWidget(self.texto_qy1)
133.
134.                self.vbox_qx2 = QtWidgets.QVBoxLayout()
135.                self.rotulo_qx2 = QtWidgets.QLabel("Qx2 (kN):")
136.                self.texto_qx2 = QtWidgets.QLineEdit()
137.                self.vbox_qx2.addWidget(self.rotulo_qx2)
138.                self.vbox_qx2.addWidget(self.texto_qx2)
139.
140.                self.vbox_qy2 = QtWidgets.QVBoxLayout()
141.                self.rotulo_qy2 = QtWidgets.QLabel("Qy2 (kN):")
142.                self.texto_qy2 = QtWidgets.QLineEdit()
143.                self.vbox_qy2.addWidget(self.rotulo_qy2)
144.                self.vbox_qy2.addWidget(self.texto_qy2)
145.
146.                self.botao_adicionar_carga_distribuida = QtWidgets.QPushButton("Adici
onar")
147.                self.botao_adicionar_carga_distribuida.setMinimumHeight(21)
148.                self.botao_adicionar_carga_distribuida.setMaximumHeight(21)
149.                self.botao_adicionar_carga_distribuida.setMinimumWidth(70)
150.                self.botao_adicionar_carga_distribuida.setMaximumWidth(70)
151.                self.botao_adicionar_carga_distribuida.clicked.connect(self.adicionar
_carga_distribuida)
152.
153.                self.tree_cargas_distribuidas = QtWidgets.QTreeWidget()
154.                self.headers = ["Barra", "Qx1", "Qx2", "Qy1", "Qy2"]
155.                self.tree_cargas_distribuidas.setColumnCount(len(self.headers))
156.                self.tree_cargas_distribuidas.setHeaderLabels(self.headers)
157.                self.tree_cargas_distribuidas.setSelectionMode(QtWidgets.QAbstractIte
mView.ExtendedSelection)
158.                self.tree_cargas_distribuidas.setIndentation(0)
159.                self.tree_cargas_distribuidas.header().resizeSection(0, 60)
160.                self.tree_cargas_distribuidas.header().resizeSection(1, 50)
161.                self.tree_cargas_distribuidas.header().resizeSection(2, 50)
```

```
162.                 self.tree_cargas_distribuidas.header().resizeSection(3, 50)
163.                 self.tree_cargas_distribuidas.header().resizeSection(4, 50)
164.
165.                 self.vbox_lista_cargas_distribuidas = QtWidgets.QVBoxLayout()
166.                 self.rotulo_lista_cargas_distribuidas = QtWidgets.QLabel("Lista de ca
rgas distribuídas:")
167.                 self.vbox_lista_cargas_distribuidas.addWidget(self.rotulo_lista_carga
s_distribuidas)
168.                 self.vbox_lista_cargas_distribuidas.addWidget(self.tree_cargas_distri
buidas)
169.
170.                 layout = QtWidgets.QGridLayout()
171.
172.                 layout.setColumnStretch(0,5)
173.                 layout.setColumnStretch(1,5)
174.                 layout.setColumnStretch(2,5)
175.                 layout.setColumnStretch(3,5)
176.
177.                 layout.setRowStretch(0,1)
178.                 layout.setRowStretch(1,1)
179.                 layout.setRowStretch(2,1)
180.                 layout.setRowStretch(3,1)
181.
182.                 layout.addLayout(self.vbox_barra,0,0,1,2)
183.                 layout.addLayout(self.vbox_qx1,1,0)
184.                 layout.addLayout(self.vbox_qx2,1,1)
185.                 layout.addLayout(self.vbox_qy1,1,2)
186.                 layout.addLayout(self.vbox_qy2,1,3)
187.                 layout.addWidget(self.botao_adicionar_carga_distribuida,2,3,1,1,QtCor
e.Qt.AlignRight)
188.                 layout.addLayout(self.vbox_lista_cargas_distribuidas,3,0,1,4)
189.
190.                 s.setLayout( layout )
191.
192.                 dock = QDockWidget("Cargas distribuídas", self)
193.                 dock.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)

194.                 dock.setWidget(s)
195.                 dock.setMinimumHeight(250)
196.                 dock.setMaximumHeight(250)
197.                 dock.setMinimumWidth(300)
198.                 dock.setMaximumWidth(300)
199.                 dock.setFloating(False)
200.                 dock.setFeatures(QDockWidget.NoDockWidgetFeatures)
201.
202.                 self.addDockWidget(Qt.RightDockWidgetArea, dock)
203.         #         self.viewMenu.addAction(dock.toggleViewAction())
204.
205.             def float_value(self, value):
206.                 try:
207.                     return float(value)
208.                 except:
209.                     return 0.0
210.
211.             def adicionar_carga_concentrada(self):
212.
213.                 idc_no = int(self.combo_no.currentText())
214.
215.                 no = self.estrutura.nos[idc_no-1]
216.
217.                 fx = self.float_value(self.texto_fx.text())
218.                 fy = self.float_value(self.texto_fy.text())
219.                 mz = self.float_value(self.texto_mz.text())
220.
221.                 self.estrutura.adicionar_carga_concentrada(idc_no, fx, fy, mz)
222.                 self.plot_canvas.desenhar_carga(no.x, no.y, fx, fy, mz)
```

```
223.
224.                 self.atualizar()
225.
226.             def adicionar_carga_distribuida(self):
227.
228.                 idc_barra = int(self.combo_barra.currentText())
229.
230.                 barra = self.estrutura.barras[idc_barra-1]
231.
232.                 qx1 = self.float_value(self.texto_qx1.text())
233.                 qx2 = self.float_value(self.texto_qx2.text())
234.                 qy1 = self.float_value(self.texto_qy1.text())
235.                 qy2 = self.float_value(self.texto_qy2.text())
236.
237.                 self.estrutura.adicionar_carga_distribuida(idc_barra, qx1, qx2, qy1,
qy2)
238.                 self.plot_canvas.desenhar_carga_distribuida2(barra.no_1.x, barra.no_1
.y, barra.no_2.x, barra.no_2.y, qx1, qx2, qy1, qy2)
239.
240.                 self.atualizar()
241.
242.             def atualizar(self):
243.
244.                 self.combo_no.clear()
245.                 self.combo_barra.clear()
246.                 self.tree_cargas_concentradas.clear()
247.                 self.tree_cargas_distribuidas.clear()
248.
249.                 for i, no in enumerate(self.estrutura.nos):
250.                     self.combo_no.addItem(str(i))
251.
252.                     for carga_concentrada in no.cargas_concentradas:
253.                         item = QtWidgets.QTreeWidgetItem(self.tree_cargas_concentrada
s)
254.                         item.setText(0,str(i))
255.                         item.setText(1,str(carga_concentrada.fx))
256.                         item.setText(2,str(carga_concentrada.fy))
257.                         item.setText(3,str(carga_concentrada.mz))
258.
259.                 for i, barra in enumerate(self.estrutura.barras):
260.                     self.combo_barra.addItem(str(i))
261.
262.                     for carga_distribuida in barra.cargas_distribuidas:
263.                         item = QtWidgets.QTreeWidgetItem(self.tree_cargas_distribuida
s)
264.                         item.setText(0,str(i))
265.                         item.setText(1,str(carga_distribuida.qx_1))
266.                         item.setText(2,str(carga_distribuida.qx_2))
267.                         item.setText(3,str(carga_distribuida.qy_1))
268.                         item.setText(4,str(carga_distribuida.qy_2))
```

B.4 Classe Painel Dados Gerais

```
1.          from PyQt5 import QtCore, QtGui, QtWidgets
2.          from PyQt5.QtCore import QDate, QFile, Qt, QTextStream
3.          from PyQt5.QtGui import (QFont, QIcon, QKeySequence, QTextCharFormat,
4.                  QTextCursor, QTextTableFormat)
5.          from PyQt5.QtPrintSupport import QPrintDialog, QPrinter
6.          from PyQt5.QtWidgets import (QWidget, QAction, QApplication, QDialog, QDockWi
dget, QTabWidget,
7.                  QFileDialog, QListWidget, QMainWindow, QMessageBox, QTextEdit)
8.
9.          class PainelDadosGerais(QMainWindow):
```

```
10.             def __init__(self, main_window):
11.                 super(PainelDadosGerais, self).__init__()
12.
13.                 self.main_window = main_window
14.                 self.estrutura = self.main_window.estrutura
15.                 self.plot_canvas = self.main_window.plot_canvas
16.
17.                 self.main_widget = QtWidgets.QWidget(self)
18.
19.                 self.createDockMateriais()
20.                 self.createDockSecoes()
21.
22.             def createDockMateriais(self):
23.
24.                 s = QWidget()
25.
26.                 self.vbox_rotulo = QtWidgets.QVBoxLayout()
27.                 self.rotulo_rotulo = QtWidgets.QLabel("Rótulo:")
28.                 self.texto_rotulo_material = QtWidgets.QLineEdit()
29.                 self.vbox_rotulo.addWidget(self.rotulo_rotulo)
30.                 self.vbox_rotulo.addWidget(self.texto_rotulo_material)
31.
32.                 self.vbox_modulo = QtWidgets.QVBoxLayout()
33.                 self.rotulo_modulo = QtWidgets.QLabel("Mód. Young (mm^2):")
34.                 self.texto_modulo = QtWidgets.QLineEdit()
35.                 self.vbox_modulo.addWidget(self.rotulo_modulo)
36.                 self.vbox_modulo.addWidget(self.texto_modulo)
37.
38.                 self.tree_materiais = QtWidgets.QTreeWidget()
39.                 self.headers = ["Descrição", "Mód. Young"]
40.                 self.tree_materiais.setColumnCount(len(self.headers))
41.                 self.tree_materiais.setHeaderLabels(self.headers)
42.                 self.tree_materiais.setSelectionMode(QtWidgets.QAbstractItemView.Exte
ndedSelection)
43.                 self.tree_materiais.setIndentation(0)
44.                 self.tree_materiais.header().resizeSection(0, 30)
45.                 self.tree_materiais.header().resizeSection(1, 40)
46.
47.                 self.vbox_lista_materiais = QtWidgets.QVBoxLayout()
48.                 self.rotulo_lista_materiais = QtWidgets.QLabel("Lista de materiais:")

49.                 self.vbox_lista_materiais.addWidget(self.rotulo_lista_materiais)
50.                 self.vbox_lista_materiais.addWidget(self.tree_materiais)
51.
52.                 self.botao_adicionar_no = QtWidgets.QPushButton("Adicionar")
53.                 self.botao_adicionar_no.setMinimumHeight(21)
54.                 self.botao_adicionar_no.setMaximumHeight(21)
55.                 self.botao_adicionar_no.setMinimumWidth(70)
56.                 self.botao_adicionar_no.setMaximumWidth(70)
57.                 self.botao_adicionar_no.clicked.connect(self.adicionar_material)
58.
59.                 layout = QtWidgets.QGridLayout()
60.
61.                 layout.setColumnStretch(0,5)
62.                 layout.setColumnStretch(1,5)
63.                 layout.setColumnStretch(2,3)
64.
65.                 layout.setRowStretch(0,1)
66.                 layout.setRowStretch(1,1)
67.                 layout.setRowStretch(2,1)
68.
69.                 layout.addLayout(self.vbox_rotulo,0,0)
70.                 layout.addLayout(self.vbox_modulo,0,1)
71.                 layout.addWidget(self.botao_adicionar_no,0,2,1,1,QtCore.Qt.AlignBotto
m)
72.                 layout.addLayout(self.vbox_lista_materiais,1,0,1,3)
```

```
73.
74.                 s.setLayout( layout )
75.
76.                 dock = QDockWidget("Materiais", self)
77.                 dock.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)

78.                 dock.setWidget(s)
79.                 dock.setMinimumHeight(180)
80.                 dock.setMaximumHeight(180)
81.                 dock.setMinimumWidth(300)
82.                 dock.setMaximumWidth(300)
83.                 dock.setFloating(False)
84.                 dock.setFeatures(QDockWidget.NoDockWidgetFeatures)
85.                 self.addDockWidget(Qt.RightDockWidgetArea, dock)
86.
87.             def adicionar_material(self):
88.                 rotulo = self.texto_rotulo_material.text()
89.                 modulo = self.texto_modulo.text()
90.                 item = QtWidgets.QTreeWidgetItem(self.tree_materiais)
91.                 item.setText(0,rotulo)
92.                 item.setText(1,modulo)
93.                 self.estrutura.adicionar_material(modulo)
94.
95.             def createDockSecoes(self):
96.
97.                 s = QWidget()
98.
99.                 self.vbox_rotulo = QtWidgets.QVBoxLayout()
100.                self.rotulo_rotulo = QtWidgets.QLabel("Rótulo:")
101.                self.texto_rotulo = QtWidgets.QLineEdit()
102.                self.vbox_rotulo.addWidget(self.rotulo_rotulo)
103.                self.vbox_rotulo.addWidget(self.texto_rotulo)
104.
105.                self.vbox_area = QtWidgets.QVBoxLayout()
106.                self.rotulo_area = QtWidgets.QLabel("Área (mm^2):")
107.                self.texto_area = QtWidgets.QLineEdit()
108.                self.vbox_area.addWidget(self.rotulo_area)
109.                self.vbox_area.addWidget(self.texto_area)
110.
111.                self.vbox_inercia = QtWidgets.QVBoxLayout()
112.                self.rotulo_inercia = QtWidgets.QLabel("Inércia (mm^4):")
113.                self.texto_inercia = QtWidgets.QLineEdit()
114.                self.vbox_inercia.addWidget(self.rotulo_inercia)
115.                self.vbox_inercia.addWidget(self.texto_inercia)
116.
117.                self.tree_secoes = QtWidgets.QTreeWidget()
118.                self.headers = ["Descrição", "Área", "Inércia"]
119.                self.tree_secoes.setColumnCount(len(self.headers))
120.                self.tree_secoes.setHeaderLabels(self.headers)
121.                self.tree_secoes.setSelectionMode(QtWidgets.QAbstractItemView.Extende
dSelection)
122.                self.tree_secoes.setIndentation(0)
123.
124.                self.vbox_lista_secoes = QtWidgets.QVBoxLayout()
125.                self.rotulo_lista_secoes = QtWidgets.QLabel("Lista de seções:")
126.                self.vbox_lista_secoes.addWidget(self.rotulo_lista_secoes)
127.                self.vbox_lista_secoes.addWidget(self.tree_secoes)
128.
129.                self.botao_adicionar_no = QtWidgets.QPushButton("Adicionar")
130.                self.botao_adicionar_no.setMinimumHeight(21)
131.                self.botao_adicionar_no.setMaximumHeight(21)
132.                self.botao_adicionar_no.setMinimumWidth(70)
133.                self.botao_adicionar_no.setMaximumWidth(70)
134.                self.botao_adicionar_no.clicked.connect(self.adicionar_secao)
135.
136.                layout = QtWidgets.QGridLayout()
```

```
137.
138.                 layout.setColumnStretch(0,5)
139.                 layout.setColumnStretch(1,5)
140.                 layout.setColumnStretch(2,3)
141.
142.                 layout.setRowStretch(0,1)
143.                 layout.setRowStretch(1,1)
144.                 layout.setRowStretch(2,1)
145.                 layout.setRowStretch(3,1)
146.
147.                 layout.addLayout(self.vbox_rotulo,0,0)
148.                 layout.addLayout(self.vbox_area,1,0)
149.                 layout.addLayout(self.vbox_inercia,1,1)
150.                 layout.addWidget(self.botao_adicionar_no,1,2,1,1,QtCore.Qt.AlignBotto
m)
151.                 layout.addLayout(self.vbox_lista_secoes,2,0,1,3)
152.
153.                 s.setLayout( layout )
154.
155.                 dock = QDockWidget("Seções", self)
156.                 dock.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)

157.                 dock.setWidget(s)
158.                 dock.setMinimumHeight(222)
159.                 dock.setMaximumHeight(222)
160.                 dock.setMinimumWidth(300)
161.                 dock.setMaximumWidth(300)
162.                 dock.setFloating(False)
163.                 dock.setFeatures(QDockWidget.NoDockWidgetFeatures)
164.
165.                 self.addDockWidget(Qt.RightDockWidgetArea, dock)
166.
167.             def adicionar_secao(self):
168.                 l = len(self.estrutura.secoes)
169.                 rotulo = self.texto_rotulo.text()
170.                 area = float(self.texto_area.text())
171.                 inercia = float(self.texto_area.text())
172.
173.                 item = QtWidgets.QTreeWidgetItem(self.tree_secoes)
174.                 item.setText(0,rotulo)
175.                 item.setText(1,str(area))
176.                 item.setText(2,str(inercia))
177.
178.                 self.estrutura.adicionar_secao(area, inercia)
```

B.5 Classe Painel Restrições

```
1.          from PyQt5 import QtCore, QtGui, QtWidgets
2.          from PyQt5.QtCore import QDate, QFile, Qt, QTextStream
3.          from PyQt5.QtGui import (QFont, QIcon, QKeySequence, QTextCharFormat, QTextCu
rsor, QTextTableFormat)
4.          from PyQt5.QtPrintSupport import QPrintDialog, QPrinter
5.          from PyQt5.QtWidgets import (QWidget, QAction, QApplication, QDialog, QDockWi
dget, QTabWidget,
6.                  QFileDialog, QListWidget, QMainWindow, QMessageBox, QTextEdit)
7.
8.          from componentes.gui.components import (ClickableLabel, ClickableLabelGroup)

9.
10.         class PainelRestricoes(QMainWindow):
11.             def __init__(self, main_window):
12.                 super(PainelRestricoes, self).__init__()
13.
14.                 self.main_window = main_window
15.                 self.estrutura = self.main_window.estrutura
```

```
16.                 self.plot_canvas = self.main_window.plot_canvas
17.
18.                 self.main_widget = QtWidgets.QWidget(self)
19.
20.                 self.createDockApoios()
21.                 self.createDockArticulacoes()
22.
23.             def createDockApoios(self):
24.
25.                 s = QWidget()
26.
27.                 self.vbox_no = QtWidgets.QVBoxLayout()
28.                 self.rotulo_no = QtWidgets.QLabel("Nó:")
29.                 self.combo_no = QtWidgets.QComboBox()
30.                 self.combo_no.setMinimumHeight(21)
31.                 self.combo_no.setMaximumHeight(21)
32.                 self.combo_no.setMinimumWidth(90)
33.                 self.combo_no.setMaximumWidth(90)
34.                 self.vbox_no.addWidget(self.rotulo_no)
35.                 self.vbox_no.addWidget(self.combo_no)
36.
37.                 self.vbox_dx = QtWidgets.QVBoxLayout()
38.                 self.rotulo_dx = QtWidgets.QLabel("Desloc. X:")
39.                 self.combo_dx = QtWidgets.QComboBox()
40.                 self.combo_dx.addItem("livre")
41.                 self.combo_dx.addItem("fixo")
42.                 self.vbox_dx.addWidget(self.rotulo_dx)
43.                 self.vbox_dx.addWidget(self.combo_dx)
44.
45.                 self.vbox_dy = QtWidgets.QVBoxLayout()
46.                 self.rotulo_dy = QtWidgets.QLabel("Desloc. Y:")
47.                 self.combo_dy = QtWidgets.QComboBox()
48.                 self.combo_dy.addItem("livre")
49.                 self.combo_dy.addItem("fixo")
50.                 self.vbox_dy.addWidget(self.rotulo_dy)
51.                 self.vbox_dy.addWidget(self.combo_dy)
52.
53.                 self.vbox_rz = QtWidgets.QVBoxLayout()
54.                 self.rotulo_rz = QtWidgets.QLabel("Rotação Z:")
55.                 self.combo_rz = QtWidgets.QComboBox()
56.                 self.combo_rz.addItem("livre")
57.                 self.combo_rz.addItem("fixo")
58.                 self.vbox_rz.addWidget(self.rotulo_rz)
59.                 self.vbox_rz.addWidget(self.combo_rz)
60.
61.                 self.botao_configurar_apoio = QtWidgets.QPushButton("Configurar")
62.                 self.botao_configurar_apoio.setMinimumHeight(21)
63.                 self.botao_configurar_apoio.setMaximumHeight(21)
64.                 self.botao_configurar_apoio.setMinimumWidth(70)
65.                 self.botao_configurar_apoio.setMaximumWidth(70)
66.                 self.botao_configurar_apoio.clicked.connect(self.configurar_apoio)
67.
68.                 layout = QtWidgets.QGridLayout()
69.
70.                 layout.setColumnStretch(0,5)
71.                 layout.setColumnStretch(1,5)
72.                 layout.setColumnStretch(2,5)
73.
74.                 layout.setRowStretch(0,1)
75.                 layout.setRowStretch(1,1)
76.                 layout.setRowStretch(2,1)
77.
78.                 layout.addLayout(self.vbox_no,0,0,1,2)
79.                 layout.addLayout(self.vbox_dx,1,0)
80.                 layout.addLayout(self.vbox_dy,1,1)
81.                 layout.addLayout(self.vbox_rz,1,2)
```

```
82.                 layout.addWidget(self.botao_configurar_apoio,2,2,1,1,QtCore.Qt.AlignR
ight)
83.
84.                 s.setLayout( layout )
85.
86.                 dock = QDockWidget("Apoios", self)
87.                 dock.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)

88.                 dock.setWidget(s)
89.                 dock.setMinimumHeight(150)
90.                 dock.setMaximumHeight(150)
91.                 dock.setMinimumWidth(300)
92.                 dock.setMaximumWidth(300)
93.                 dock.setFloating(False)
94.                 dock.setFeatures(QDockWidget.NoDockWidgetFeatures)
95.                 self.addDockWidget(Qt.RightDockWidgetArea, dock)
96.
97.
98.             def createDockArticulacoes(self):
99.
100.                s = QWidget()
101.
102.                self.vbox_no1 = QtWidgets.QVBoxLayout()
103.                self.rotulo_no1 = QtWidgets.QLabel("Nó:")
104.                self.combo_no1 = QtWidgets.QComboBox()
105.                self.vbox_no1.addWidget(self.rotulo_no1)
106.                self.vbox_no1.addWidget(self.combo_no1)
107.
108.                self.vbox_tipo_no = QtWidgets.QVBoxLayout()
109.                self.rotulo_tipo_no = QtWidgets.QLabel("Tipo do nó:")
110.                self.combo_tipo_no = QtWidgets.QComboBox()
111.                self.combo_tipo_no.addItem("livre")
112.                self.combo_tipo_no.addItem("fixo")
113.                self.vbox_tipo_no.addWidget(self.rotulo_tipo_no)
114.                self.vbox_tipo_no.addWidget(self.combo_tipo_no)
115.
116.                self.botao_configurar_no = QtWidgets.QPushButton("Configurar nó")
117.                self.botao_configurar_no.setMinimumHeight(21)
118.                self.botao_configurar_no.setMaximumHeight(21)
119.                self.botao_configurar_no.setMinimumWidth(70)
120.                self.botao_configurar_no.setMaximumWidth(70)
121.                self.botao_configurar_no.clicked.connect(self.botao_configurar_no)
122.
123.                self.vbox_barra = QtWidgets.QVBoxLayout()
124.                self.rotulo_barra = QtWidgets.QLabel("Barra:")
125.                self.combo_barra = QtWidgets.QComboBox()
126.                self.vbox_barra.addWidget(self.rotulo_barra)
127.                self.vbox_barra.addWidget(self.combo_barra)
128.
129.                self.vbox_tipo_barra = QtWidgets.QVBoxLayout()
130.                self.rotulo_tipo_barra = QtWidgets.QLabel("Tipo da barra:")
131.                self.combo_tipo_barra = QtWidgets.QComboBox()
132.                self.vbox_tipo_barra.addWidget(self.rotulo_tipo_barra)
133.                self.vbox_tipo_barra.addWidget(self.combo_tipo_barra)
134.
135.                self.botao_configurar_barra = QtWidgets.QPushButton("Configurar barra
")
136.                self.botao_configurar_barra.setMinimumHeight(21)
137.                self.botao_configurar_barra.setMaximumHeight(21)
138.                self.botao_configurar_barra.setMinimumWidth(70)
139.                self.botao_configurar_barra.setMaximumWidth(70)
140.                self.botao_configurar_barra.clicked.connect(self.botao_configurar_bar
ra)
141.
142.                layout2 = QtWidgets.QGridLayout()
143.
```

```
144.                 layout2.setColumnStretch(0,5)
145.                 layout2.setColumnStretch(1,5)
146.
147.                 layout2.setRowStretch(0,1)
148.                 layout2.setRowStretch(1,1)
149.                 layout2.setRowStretch(2,1)
150.                 layout2.setRowStretch(3,1)
151.
152.                 layout2.addLayout(self.vbox_no_inicial,0,0)
153.                 layout2.addLayout(self.vbox_no_final,0,1)
154.                 layout2.addLayout(self.vbox_material,1,0,1,1)
155.                 layout2.addLayout(self.vbox_secao,1,1,1,1)
156.                 layout2.addWidget(self.botao_adicionar_barra,2,0,1,2,QtCore.Qt.AlignB
ottom)
157.
158.                 s.setLayout( layout2 )
159.
160.                 dock = QDockWidget("Articulações", self)
161.                 dock.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)

162.                 dock.setWidget(s)
163.                 dock.setMinimumHeight(250)
164.                 dock.setMaximumHeight(250)
165.                 dock.setMinimumWidth(300)
166.                 dock.setMaximumWidth(300)
167.                 dock.setFloating(False)
168.                 dock.setFeatures(QDockWidget.NoDockWidgetFeatures)
169.
170.                 self.addDockWidget(Qt.RightDockWidgetArea, dock)
171.         #        self.viewMenu.addAction(dock.toggleViewAction())
172.
173.             def configurar_apoio(self):
174.                 r = len(self.estrutura.nos)
175.                 coord_x = float(self.texto_coordenada_x.text())
176.                 coord_y = float(self.texto_coordenada_y.text())
177.
178.                 self.estrutura.adicionar_no(coord_x,coord_y)
179.                 self.plot_canvas.desenhar_no(r,coord_x,coord_y)
180.
181.         #        self.atualizar()
182.
183.             def adicionar_barra(self):
184.
185.                 idc = len(self.estrutura.barras)
186.
187.                 idc_no_inicial = int(self.combo_no_inicial.currentText())
188.                 idc_no_final = int(self.combo_no_final.currentText())
189.
190.                 no_inicial = self.estrutura.nos[idc_no_inicial]
191.                 no_final = self.estrutura.nos[idc_no_final]
192.
193.                 self.estrutura.adicionar_barra(idc_no_inicial+1,idc_no_final+1,1,1)

194.                 self.plot_canvas.desenhar_barra(idc, no_inicial.x, no_inicial.y, no_f
inal.x, no_final.y)
195.
196.         #       self.atualizar()
197.
198.             def atualizar(self):
199.
200.                 self.combo_no.clear()
201.
202.                 for i, no in enumerate(self.estrutura.nos):
203.                     print('testando')
204.                     self.combo_no.addItem(str(i))
```

B.6 Classe Painel Estrutura

```
1.          from PyQt5 import QtCore, QtGui, QtWidgets
2.          from PyQt5.QtCore import QDate, QFile, Qt, QTextStream
3.          from PyQt5.QtGui import (QFont, QIcon, QKeySequence, QTextCharFormat,
4.                  QTextCursor, QTextTableFormat)
5.          from PyQt5.QtPrintSupport import QPrintDialog, QPrinter
6.          from PyQt5.QtWidgets import (QWidget, QAction, QApplication, QDialog, QDockWi
dget, QTabWidget,
7.                  QFileDialog, QListWidget, QMainWindow, QMessageBox, QTextEdit)
8.
9.          from componentes.gui.components import (ClickableLabel, ClickableLabelGroup)

10.
11.         class PainelEstrutura(QMainWindow):
12.             def __init__(self, main_window):
13.                 super(PainelEstrutura, self).__init__()
14.
15.                 self.main_window = main_window
16.                 self.estrutura = self.main_window.estrutura
17.                 self.plot_canvas = self.main_window.plot_canvas
18.
19.                 self.main_widget = QtWidgets.QWidget(self)
20.
21.                 self.createDockNos()
22.                 self.createDockBarras()
23.
24.             def createDockNos(self):
25.
26.                 s = QWidget()
27.
28.                 self.vbox_coordenada_x = QtWidgets.QVBoxLayout()
29.                 self.rotulo_coordenada_x = QtWidgets.QLabel("Coordenada X (m):")
30.                 self.texto_coordenada_x = QtWidgets.QLineEdit()
31.                 self.vbox_coordenada_x.addWidget(self.rotulo_coordenada_x)
32.                 self.vbox_coordenada_x.addWidget(self.texto_coordenada_x)
33.
34.                 self.vbox_coordenada_y = QtWidgets.QVBoxLayout()
35.                 self.rotulo_coordenada_y = QtWidgets.QLabel("Coordenada Y (m):")
36.                 self.texto_coordenada_y = QtWidgets.QLineEdit()
37.                 self.vbox_coordenada_y.addWidget(self.rotulo_coordenada_y)
38.                 self.vbox_coordenada_y.addWidget(self.texto_coordenada_y)
39.
40.                 self.botao_adicionar_no = QtWidgets.QPushButton("Adicionar")
41.                 self.botao_adicionar_no.setMinimumHeight(21)
42.                 self.botao_adicionar_no.setMaximumHeight(21)
43.                 self.botao_adicionar_no.setMinimumWidth(70)
44.                 self.botao_adicionar_no.setMaximumWidth(70)
45.                 self.botao_adicionar_no.clicked.connect(self.adicionar_no)
46.
47.                 self.tree_nos = QtWidgets.QTreeWidget()
48.                 self.headers = ["Id.", "Coord. X", "Coord. Y"]
49.                 self.tree_nos.setColumnCount(len(self.headers))
50.                 self.tree_nos.setHeaderLabels(self.headers)
51.                 self.tree_nos.setSelectionMode(QtWidgets.QAbstractItemView.ExtendedSe
lection)
52.                 self.tree_nos.setIndentation(0)
53.                 self.tree_nos.header().resizeSection(0, 50)
54.                 self.tree_nos.header().resizeSection(1, 70)
55.                 self.tree_nos.header().resizeSection(2, 70)
56.
57.                 self.vbox_lista_nos = QtWidgets.QVBoxLayout()
58.                 self.rotulo_lista_nos = QtWidgets.QLabel("Lista de nós:")
59.                 self.vbox_lista_nos.addWidget(self.rotulo_lista_nos)
```

```
60.                 self.vbox_lista_nos.addWidget(self.tree_nos)
61.
62.
63.                 layout1 = QtWidgets.QGridLayout()
64.
65.                 layout1.setColumnStretch(0,5)
66.                 layout1.setColumnStretch(1,5)
67.                 layout1.setColumnStretch(2,3)
68.
69.                 layout1.setRowStretch(0,1)
70.                 layout1.setRowStretch(1,1)
71.                 layout1.setRowStretch(2,1)
72.
73.                 layout1.addLayout(self.vbox_coordenada_x,0,0)
74.                 layout1.addLayout(self.vbox_coordenada_y,0,1)
75.                 layout1.addWidget(self.botao_adicionar_no,0,2,1,1,QtCore.Qt.AlignBott
om)
76.                 layout1.addLayout(self.vbox_lista_nos,1,0,1,3)
77.
78.                 s.setLayout( layout1 )
79.
80.                 dock = QDockWidget("Nós", self)
81.                 dock.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)

82.                 dock.setWidget(s)
83.                 dock.setMinimumHeight(240)
84.                 dock.setMaximumHeight(240)
85.                 dock.setMinimumWidth(300)
86.                 dock.setMaximumWidth(300)
87.                 dock.setFloating(False)
88.                 dock.setFeatures(QDockWidget.NoDockWidgetFeatures)
89.
90.                 self.addDockWidget(Qt.RightDockWidgetArea, dock)
91.
92.             def createDockBarras(self):
93.
94.                 s = QWidget()
95.
96.                 self.vbox_no_inicial = QtWidgets.QVBoxLayout()
97.                 self.rotulo_no_inicial = QtWidgets.QLabel("Nó inicial:")
98.                 self.combo_no_inicial = QtWidgets.QComboBox()
99.                 self.vbox_no_inicial.addWidget(self.rotulo_no_inicial)
100.                self.vbox_no_inicial.addWidget(self.combo_no_inicial)
101.
102.                self.vbox_no_final = QtWidgets.QVBoxLayout()
103.                self.rotulo_no_final = QtWidgets.QLabel("Nó final:")
104.                self.combo_no_final = QtWidgets.QComboBox()
105.                self.vbox_no_final.addWidget(self.rotulo_no_final)
106.                self.vbox_no_final.addWidget(self.combo_no_final)
107.
108.                self.vbox_material = QtWidgets.QVBoxLayout()
109.                self.rotulo_material = QtWidgets.QLabel("Material:")
110.                self.combo_material = QtWidgets.QComboBox()
111.                self.vbox_material.addWidget(self.rotulo_material)
112.                self.vbox_material.addWidget(self.combo_material)
113.
114.                self.vbox_secao = QtWidgets.QVBoxLayout()
115.                self.rotulo_secao = QtWidgets.QLabel("Seção:")
116.                self.combo_secao = QtWidgets.QComboBox()
117.                self.vbox_secao.addWidget(self.rotulo_secao)
118.                self.vbox_secao.addWidget(self.combo_secao)
119.
120.                self.botao_adicionar_barra = QtWidgets.QPushButton("Adicionar")
121.                self.botao_adicionar_barra.setMinimumHeight(21)
122.                self.botao_adicionar_barra.setMaximumHeight(21)
123.                self.botao_adicionar_barra.setMinimumWidth(70)
```

```
124.                 self.botao_adicionar_barra.setMaximumWidth(70)
125.                 self.botao_adicionar_barra.clicked.connect(self.adicionar_barra)
126.
127.                 self.tree_barras = QtWidgets.QTreeWidget()
128.                 self.headers = ["Id.", "Nó 1", "Nó 2"]
129.                 self.tree_barras.setColumnCount(len(self.headers))
130.                 self.tree_barras.setHeaderLabels(self.headers)
131.                 self.tree_barras.setSelectionMode(QtWidgets.QAbstractItemView.Extende
dSelection)
132.                 self.tree_barras.setIndentation(0)
133.
134.                 self.vbox_lista_barras = QtWidgets.QVBoxLayout()
135.                 self.rotulo_lista_barras = QtWidgets.QLabel("Lista de barras:")
136.                 self.vbox_lista_barras.addWidget(self.rotulo_lista_barras)
137.                 self.vbox_lista_barras.addWidget(self.tree_barras)
138.
139.                 layout2 = QtWidgets.QGridLayout()
140.
141.                 layout2.setColumnStretch(0,5)
142.                 layout2.setColumnStretch(1,5)
143.
144.                 layout2.setRowStretch(0,1)
145.                 layout2.setRowStretch(1,1)
146.                 layout2.setRowStretch(2,1)
147.                 layout2.setRowStretch(3,1)
148.                 layout2.setRowStretch(4,5)
149.
150.                 layout2.addLayout(self.vbox_no_inicial,0,0)
151.                 layout2.addLayout(self.vbox_no_final,0,1)
152.                 layout2.addLayout(self.vbox_material,1,0,1,1)
153.                 layout2.addLayout(self.vbox_secao,1,1,1,1)
154.                 layout2.addWidget(self.botao_adicionar_barra,2,0,1,2,QtCore.Qt.AlignB
ottom)
155.                 layout2.addLayout(self.vbox_lista_barras,3,0,1,2)
156.
157.                 s.setLayout( layout2 )
158.
159.                 dock = QDockWidget("Barras", self)
160.                 dock.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)

161.                 dock.setWidget(s)
162.                 dock.setMinimumHeight(250)
163.                 dock.setMaximumHeight(250)
164.                 dock.setMinimumWidth(300)
165.                 dock.setMaximumWidth(300)
166.                 dock.setFloating(False)
167.                 dock.setFeatures(QDockWidget.NoDockWidgetFeatures)
168.
169.                 self.addDockWidget(Qt.RightDockWidgetArea, dock)
170.
171.             def adicionar_no(self):
172.                 r = len(self.estrutura.nos)
173.                 coord_x = float(self.texto_coordenada_x.text())
174.                 coord_y = float(self.texto_coordenada_y.text())
175.
176.                 self.estrutura.adicionar_no(coord_x,coord_y)
177.                 self.plot_canvas.desenhar_no(r,coord_x,coord_y)
178.
179.                 self.atualizar()
180.
181.             def adicionar_barra(self):
182.
183.                 idc = len(self.estrutura.barras)
184.
185.                 idc_no_inicial = int(self.combo_no_inicial.currentText())
186.                 idc_no_final = int(self.combo_no_final.currentText())
```

```
187.
188.                 no_inicial = self.estrutura.nos[idc_no_inicial]
189.                 no_final = self.estrutura.nos[idc_no_final]
190.
191.                 self.estrutura.adicionar_barra(idc_no_inicial+1,idc_no_final+1,1,1)

192.                 self.plot_canvas.desenhar_barra(idc, no_inicial.x, no_inicial.y, no_f
inal.x, no_final.y)
193.
194.                 self.atualizar()
195.
196.             def atualizar(self):
197.
198.                 self.combo_no_inicial.clear()
199.                 self.combo_no_final.clear()
200.                 self.combo_material.clear()
201.                 self.combo_secao.clear()
202.                 self.tree_nos.clear()
203.                 self.tree_barras.clear()
204.
205.                 for i, secao in enumerate(self.estrutura.secoes):
206.                     self.combo_secao.addItem(str(i))
207.
208.                 for i, material in enumerate(self.estrutura.materiais):
209.                     self.combo_material.addItem(str(i))
210.
211.                 for i, no in enumerate(self.estrutura.nos):
212.                     self.combo_no_inicial.addItem(str(i))
213.                     self.combo_no_final.addItem(str(i))
214.
215.                     item = QtWidgets.QTreeWidgetItem(self.tree_nos)
216.                     item.setText(0,str(i))
217.                     item.setText(1,str(no.x))
218.                     item.setText(2,str(no.y))
219.
220.                 for i, barra in enumerate(self.estrutura.barras):
221.                     item = QtWidgets.QTreeWidgetItem(self.tree_barras)
222.                     item.setText(0,str(i))
223.                     item.setText(1,str(self.estrutura.nos.index(barra.no_1)))
224.                     item.setText(2,str(self.estrutura.nos.index(barra.no_2)))
```