UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

THIAGO SEIJI ENOKIDA

DESENVOLVIMENTO DE CÓDIGO COMPUTACIONAL PARA ANÁLISE LINEAR ESTÁTICA DE VIGAS RETANGULARES DE ALTURA VARIÁVEL PELO MÉTODO DOS ELEMENTOS FINITOS

CAMPO MOURÃO

2021

THIAGO SEIJI ENOKIDA

DESENVOLVIMENTO DE CÓDIGO COMPUTACIONAL PARA ANÁLISE LINEAR ESTÁTICA DE VIGAS RETANGULARES DE ALTURA VARIÁVEL PELO MÉTODO DOS ELEMENTOS FINITOS

Development of computational code for linear static analysis of rectangular beams with variable height by the Finite Element Method

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Engenharia Civil da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Leandro Waidemam

CAMPO MOURÃO

2021

THIAGO SEIJI ENOKIDA

DESENVOLVIMENTO DE CÓDIGO COMPUTACIONAL PARA ANÁLISE LINEAR ESTÁTICA DE VIGAS RETANGULARES DE ALTURA VARIÁVEL PELO MÉTODO DOS ELEMENTOS FINITOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Civil da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 01 de dezembro de 2021

_____

Leandro Waidemam
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Marcelo Rodrigo Carreira
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Ronaldo Rigobello
Doutorado
Universidade Tecnológica Federal do Paraná

CAMPO MOURÃO

2021

AGRADECIMENTOS

Dedico este trabalho à minha família, principalmente meu pai, Milton, e minha mãe, Izabel, e seus sacrifícios para que isso fosse possível, e minha irmã, que sempre me deu suporte e forças para continuar.

Agradeço ao meu amigo, Devair, pela companhia e motivação em momentos tensos de prazos apertados, e pelo café que sempre me dava o empurrão que precisava para terminar o dia concentrado.

Agradeço a minha namorada, Rebeca, que compartilhou comigo as conquistas e frustrações ao longo de todo o curso, me dando alívio e calma nos momentos que precisava.

Agradeço ao meu orientador Prof. Dr. Leandro Waidemam, que fez crescer em mim o gosto pela área de engenharia de estruturas, e pela atenção, bom humor e gentileza que me foi dado durante as aulas e atendimentos.

Agradeço de mesma maneira aos professores Dr. Ronaldo Rigobello, Dr. Jorge Góes, Dr. Marcelo Carreira, Me. Angelo Giovanni e Dr. Jeferson Rafael pelas matérias mais agradáveis e proveitosas que tive, reforçando em mim ainda mais o gosto pela engenharia.

Por fim, dedico a Agnaldo Malaquias (in memorian), que teve importante participação na minha formação como pessoa, e por ter um ótimo mentor, conselheiro e amigo. Seu carinho e ensinamentos vivem e viverão em mim para toda a vida.

RESUMO

Este trabalho tem como objetivo apresentar uma formulação baseada no Método dos Elementos Finitos e um código computacional em linguagem Python para a análise elástica-linear de vigas de Euler-Bernoulli de altura variável, sob diferentes condições de carregamentos e vinculações. A discretização estrutural é feita utilizando elementos finitos com dois nós, cada um com dois graus de liberdade: deslocamento vertical e giro nodal. Para a aproximação dos deslocamentos ao longo do elemento utiliza-se uma função polinomial de terceiro grau. A fim de validar o código computacional desenvolvido, ao final do trabalho são apresentados exemplos numéricos de alguns elementos estruturais, sendo os resultados comparados com os fornecidos por outros autores e por outros softwares. Os resultados indicam que a formulação apresentada e o software desenvolvido fornecem soluções precisas e confiáveis para o problema em questão.

Palavras-chave: Método dos Elementos Finitos; análise elástica; vigas; Python.

ABSTRACT

This work aims to present a finite element formulation and a computational code in Python language for elastic-linear analysis of Euler-Bernoulli beams of variable height under different loading and supports conditions. Structural discretization is done using finite elements with two nodes, each one with two degrees of freedom: vertical displacement and nodal rotation. To aproximate the displacements along the element, a polynomial function of third degree is used. To validate the computational code developed, at the end of the work, numerical examples of some structural elements are presented, and the results are compared with those provided by other authors and other softwares. The results shows that the presented formulation and the developed software provide accurate and reliable solutions for the problem in question.

Keywords: Finite Element Method; elastic analysis; beams; Python.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SÍMBOLOS

U_e^*	Trabalho virtual externo
U_i^*	Trabalho virtual interno
F	Força vertical
Δ^*	Deslocamento nodal virtual
M	Momento
ϕ^*	Giro nodal virtual
$q(x)$	Carregamento linearmente distribuído
$\Delta(\mathrm{x})$	Deslocamento virtual
σ	Tensão normal
$\delta\varepsilon^*$	Variação da deformação normal virtual
ε	Deformação normal
y	Distância do centroide da seção transversal até a fibra analisada
v	Deslocamento transversal
E	Módulo de elasticidade longitudinal
δv^*	Variação dado deslocamento transversal virtual
h	Altura da seção transversal
b	Base da seção transversal
I	Momento de inércia da seção transversal em relação ao eixo neutro
h_1	Altura inicial da seção transversal
h_2	Altura final da seção transversal
L	Comprimento do elemento finito
θ	Inclinação da seção transversal
$\delta\theta^*$	Variação da inclinação virtual da seção transversal
$[k_e]$	Matriz de rigidez elemental em coordenadas locais
$\{\delta_e\}$	Vetor de deslocamentos nodais em coordenadas locais
$\{F_e\}$	Vetor de forças nodais em coordenadas locais

SUMÁRIO

1 INTRODUÇÃO

A engenharia civil é a ciência que transforma e adapta a natureza, a fim de otimizar a qualidade de vida das pessoas, se utilizando de ferramentas como a física, química e matemática, além de economia e gestão de pessoas. Um dos ramos da engenharia civil é a engenharia estrutural, que por meio de análises estáticas e dinâmicas se dedica primariamente à análise e dimensionamento de estruturas, equilibrando economia com eficiência e segurança.

Dentre os elementos que compõem os sistemas estruturais, pode-se citar as vigas, elementos estruturais sujeitos a carregamentos transversais, comumente utilizados para receber esforços resultantes de lajes, podendo ser composta de concreto, aço ou madeira.

Uma das medidas a serem estudadas nas vigas são os momentos de inércia, que medem a distribuição da massa de um corpo em torno de seu eixo de rotação, sendo inteiramente relacionadas a área da seção e sua geometria, e a variação da seção, que causa uma variação no momento de inércia tem sido utilizado extensivamente em pontes há mais de 50 anos, e também nos últimos tempos, aumentado o uso deste tipo de solução estrutural em edifícios industriais e comerciais.

Através do tempo, diversos métodos foram desenvolvidos visando descrever os esforços que atuam nos elementos estruturais. De acordo com Logan (2007) o desenvolvimento moderno do Método dos Elementos Finitos começou em 1940 no campo da engenharia estrutural pelo trabalho de Hrennikoff em 1941 e McHenry em 1943, com a necessidade de resolver problemas complexos de elasticidade e análise estrutural para a engenharia civil e aeroespacial, porém havia dificuldades na utilização do método devido aos extensos cálculos a serem realizados.

Com o advento dos computadores e o crescente poder computacional na década de 70, o MEF se tornou um método mais atraente, devido a sua fácil tradução para linguagem computacional. Agora, amplamente estudado, o MEF realiza análises complexas onde existe a limitação nos métodos clássicos.

Este trabalho visa empregar o poder de processamento matemático do computador como solução para cálculos extensos, avaliando o comportamento estrutural de vigas onde ocorre variação linear na altura da seção transversal ao longo do vão, se utilizando do modelo de Euler-Bernoulli, o Princípio dos trabalhos virtuais e o MEF para elaboração e implementação do código computacional.

2 OBJETIVOS

2.1 Objetivo geral

Este trabalho visa avaliar o comportamento elástico linear de vigas retangulares com variação linear de altura no seu comprimento, submetidas a variados esforços e condições de contorno utilizando como ferramenta um código computacional implementado a partir do Método dos Elementos Finitos

2.2 Objetivos específicos

- Compreender a formulação teórica de um modelo de análise elástica linear de vigas, com base na teoria de Euler-Bernoulli e no Métodos dos Elementos Finitos;
- Elaborar e validar uma rotina computacional em linguagem Python que realize a análise estrutural proposta;
- Analisar o comportamento elástico linear de vigas retangulares com variação linear de altura no seu comprimento.

3 JUSTIFICATIVA

As vigas são elementos estruturais presentes no sistema laje-viga-pilar, tem como função principal o transporte das cargas, recebendo-as da laje e transmitindo aos pilares. São elementos lineares sujeitos principalmente a esforços de momento fletor e força cortante, podendo ser solicitadas também a força normal e momento torçor. A análise do seu comportamento mecânico possibilita ao correto dimensionamento estrutural, de forma que haja economia de materiais e possibilitando de maneira conjunta, garantia da segurança e conforto dos usuários da estrutura.

As vigas de altura variável se apresentam como uma solução econômica e estética, a fim de otimizar o projeto de estrutura de grandes vãos e acrescentar elementos estéticos com senso de profundidade, sendo usualmente aplicadas como uma solução em elementos pré-moldados. Comumente utilizadas como elementos de pontes e consolos, otimiza o uso de materiais, e possibilita também uma estética agradável e uma flexibilidade na gestão do ambiente, no ponto de vista da arquitetura.

Duas teorias são amplamente empregadas para se descrever o comportamento desses elementos, a teoria de Euler-Bernoulli, e a teoria de Timoshenko. Dentro do presente projeto, utiliza-se as formulações propostas por Euler-Bernoulli, mesmo ocorrendo uma variação da altura, a deformação devida à cortante é pequena o suficiente para que a formulação Euler-Bernoulli apresente resultados satisfatórios. É o caso dos tipos mais usuais de vigas, nas quais a relação altura da seção transversal e o vão é pequena.

De forma a alcançar um dos objetivos deste trabalho, o de implementar uma rotina computacional para a análise estrutural proposta, optou-se por utilizar o MEF, uma ferramenta matemática poderosa desenvolvida no início da década de 1940 e popularizada apenas nos anos de 1970, com o advento dos computadores e o poder de processamento matemático apresentado por eles. A sua maneira de transformar a solução das equações diferenciais que descrevem os problemas físicos na solução de um sistema de equações lineares facilmente calculado pelo computador, torna o método flexível e facilmente programável, além de fornecer soluções precisas de forma eficiente.

4 REVISÃO BIBLIOGRÁFICA

4.1 Vigas

Geralmente as vigas são os elementos estruturais responsáveis por transmitir as cargas verticais das lajes e as distribuídas aplicadas na própria viga, para os pilares (CORRÊA, 2013). São estruturas lineares, podendo ser dispostas horizontalmente ou inclinadas, com um ou mais apoios (móvel ou fixo), engastes etc. de tal forma a garantir que tais barras sejam no mínimo isostáticas. Podem ser confeccionadas em madeira, aço, ferro fundido, concreto (armado ou protendido) e alumínio, com aplicações nos mais diversos tipos de construções (SOUZA; RODRIGUES; MASCIA, 2008).

Segundo Beer, Johnston e Mazurek (2012), como elemento estrutural linear entende-se aquele que possui uma dimensão (comprimento) muito superior às demais dimensões (dimensões transversais). No caso das vigas o carregamento atuante é perpendicular ao seu eixo longitudinal.

São consideradas vigas de altura variável aquelas em que a altura varia num mesmo vão, formando trapézios ou curvas. Tal configuração permite otimizar o projeto de estruturas com grandes vãos, podendo ser uma solução econômica, além de garantir o bom funcionamento da estrutura. São frequentemente adotadas como solução em obras de arte, como viadutos e pontes, e comumente encontradas em obras com elementos pré-moldados (CORRÊA, 2013). A figura 1 demonstra uma situação de uso da viga de altura variável.

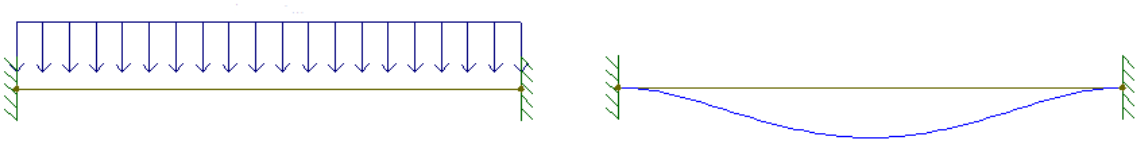
Figura 1 – Viga de altura variável aplicada em estruturas de pontes

Fonte: Moura (2021, sem paginação)

Várias vantagens podem ser alcançadas com a variação de seção transversal, tanto a nível arquitetônico, construtivo e econômico. A nível arquitetônico é possível criar formas e espaços amplos, além de dispensar a moldagem in loco, tornando necessário apenas o transporte e montagem de elementos pré-moldados. A nível econômico permite que zonas de elevado esforço solicitante possuam seções maiores, enquanto zonas com menor esforço se despende de menos material para a seção resistente, permitindo uma redução significativa na quantidade de material despendido, otimizando a relação custo/benefício, sem comprometer a estabilidade e a segurança da estrutura.

As deformações excessivas apresentadas pelos elementos estruturais podem causar desconforto aos usuários, como também podem alterar a aparência e a eficiência de uma estrutura. No entanto, as mais severas consequências são devidas aos danos locais, que se apresentam como fissuração de elementos estruturais e não-estruturais ou rotação excessiva (LIMA; FONTES; LIMA, 2003). A figura 2 exemplifica a configuração deformada que ocorre em uma viga bi-engastada solicitada por um carregamento uniformemente distribuído

Figura 2 – Viga bi engastada e sua configuração deformada



Fonte: Autoria própria (2021)

A ABNT NBR 8681:2004 define os padrões normativos para ações e seguranças nas estruturas e determina como requisitos gerais que a estrutura atenda a dois requisitos de estados limites, que dependem dos tipos de materiais de construção empregados. O primeiro destes são os estados limites últimos, que são um estado onde, por sua ocorrência, determinam a paralisação ao todo ou em partes, do uso da construção, e são caracterizados por:

1. Perda de equilíbrio, global ou parcial, admitida a estrutura como um corpo rígido;
2. Ruptura ou deformação plástica excessiva dos materiais;
3. Transformação da estrutura, no todo ou em parte, em sistema hipostático;
4. Instabilidade por deformação;

5. Instabilidade dinâmica.

O segundo são os estados limites de serviço, estados que causam efeitos estruturais que não respeitam as condições especificadas para o uso normal da construção, demonstrando indícios de comprometimento da durabilidade das estruturas, caracterizados por:

1. Danos ligeiros ou localizados, que comprometam o aspecto estético da construção ou a durabilidade da estrutura;
2. Deformações excessivas que afetem a utilização normal da construção ou seu aspecto estético;
3. Vibração excessiva ou desconfortável.

Os valores limites dos estados limites dependem do material a ser empregado no elemento estrutural, sendo as mais usuais a ABNT NBR 8800:2008 para elementos de aço, a ABNT NBR 6118:2014 para elementos de concreto, e a ABNT NBR 7190:1997 para elementos de madeira.

4.2 Modelo de Euler-Bernoulli

A análise das deflexões de vigas é bastante comum em problemas de engenharia, tornando-se fundamental o seu estudo. Para esta finalidade existem algumas teorias, podendo-se destacar os modelos de vigas de Euler-Bernoulli e de Timoshenko. A principal diferença entre os dois modelos está relacionada ao fato da não consideração da deformação de cisalhamento presentes nas seções transversais no modelo de vigas de Euler-Bernoulli (BITTENCOURT; FEIJÓO, 1999).

Segundo Han, Benoroya e Wei (1999), as seguintes hipóteses físicas são adotadas para o modelo de Euler-Bernoulli:

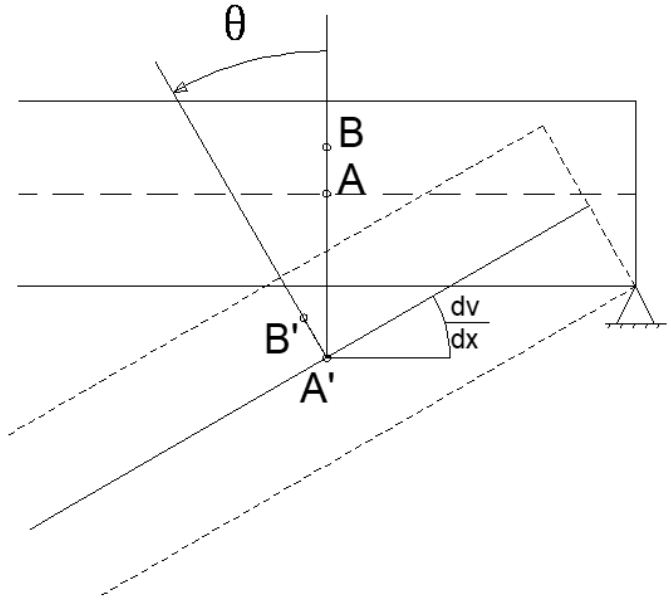
1. Uma dimensão é consideravelmente maior que as outras duas;
2. O material é linearmente elástico;
3. O coeficiente de Poisson é negligenciável;
4. A área da seção transversal é simétrica para que a linha neutra e o centroide coincidam;

5. O plano perpendicular a linha neutra permanece perpendicular após a deformação;
6. O ângulo de rotação da seção transversal é pequeno;
7. As deformações por cisalhamento são desprezadas.

A hipótese admitida pela teoria de Euler-Bernoulli de que havendo uma pequena deformação, as seções transversais permanecem planas e perpendiculares a linha neutra, é adotada para o presente trabalho. Caso não fosse, a teoria de Timoshenko seria a mais adequada.

Desta maneira, a figura 3 ilustra os deslocamentos e giros de seção transversal considerados no modelo.

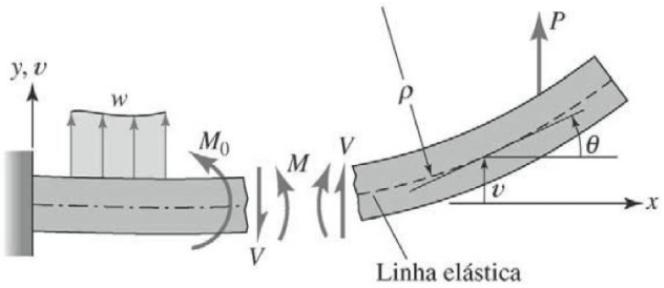
Figura 3 – Modelo de deslocamentos e giros de seção transversal



Fonte: Autoria própria (2021)

Desta maneira, a partir da viga reta deslocada, é possível se desenvolver a equação diferencial da linha elástica, considerando uma viga engastada e carregada, conforme apresenta a figura 4.

Figura 4 – Cargas positivas e resultantes das forças internas.



Fonte: Ugural (2009, p. 414)

Considerando a origem das coordenadas na extremidade engastada, admite-se que o plano *xy* é o plano de flexão. O deslocamento v de um ponto A qualquer da viga a uma distância x da origem é o deslocamento do ponto na direção y, medido do eixo x até a linha elástica. (UGURAL, 2009).

De maneira analítica, a definição da curvatura k a uma distância x do eixo y é expressa da seguinte maneira:

$$k = \frac{1}{\rho} = \frac{\frac{d^2v}{dx^2}}{\left[1 + \left(\frac{dv}{dx}\right)^2\right]^{\frac{3}{2}}} \tag{1}$$

Considerando a primeira hipótese fundamental da teoria de Euler-Bernoulli para vigas esbeltas. Os deslocamentos do eixo da viga são pequenos quando comparados com o vão da viga. Os ângulos de rotação da curva de deslocamentos também são muito pequenos e aproximadamente iguais a inclinação, $\theta = dv/dx$ (UGURAL, 2009 p. 239)

Assim, o quadrado da inclinação pode ser considerado desprezível e a equação (1) pode ser reescrita como segue:

$$k = \frac{1}{\rho} = \frac{d^2v}{dx^2} \tag{2}$$

Assim, k representa a taxa com a qual a inclinação varia ao longo da viga. Para uma viga elástica linear cuja seção transversal é simétrica em relação ao plano

de carregamento, a curvatura está relacionada ao momento, através da seguinte expressão, conforme Ugural (2009):

$$\frac{d^2v}{dx^2} = \frac{M}{EI(x)} \tag{3}$$

Considerando as relações diferenciais entre carregamento e força cortante e entre força cortante e momento fletor (BEER; JOHNSTON; MAZUREK, 2012) pode-se ainda escrever as relações descritas em (4).

$$\begin{aligned} EI(x)\frac{d^2v}{dx^2} &= M(x) \\ \frac{d}{dx}\left[EI(x)\frac{d^2v}{dx^2}\right] &= V(x) \\ \frac{d^2}{dx^2}\left[EI(x)\frac{d^2v}{dx^2}\right] &= -w(x) \end{aligned} \tag{4}$$

4.3 Método dos Elementos Finitos

Estruturas, em geral, são constituídas de um conjunto de diversos componentes, admitindo geometrias complexas, carregamentos e diferentes propriedades dos materiais. Os métodos clássicos admitem certos níveis de simplificação, o que em alguns casos não é o adequado. Quando se trata de análise de estruturas complexas, deve-se recorrer a procedimentos mais gerais. Dentre os métodos de análise a mais amplamente utilizada é o método dos elementos finitos. A análise por elementos finitos é um procedimento numérico bem adequado para implementação em computadores digitais (UGURAL, 2009).

O método consiste na formulação de um conjunto simultâneo de equações algébricas que buscam relacionar as forças aos correspondentes deslocamentos em pontos selecionados ou “nós”.

Segundo Ugural (2009), o método oferece muitas vantagens:

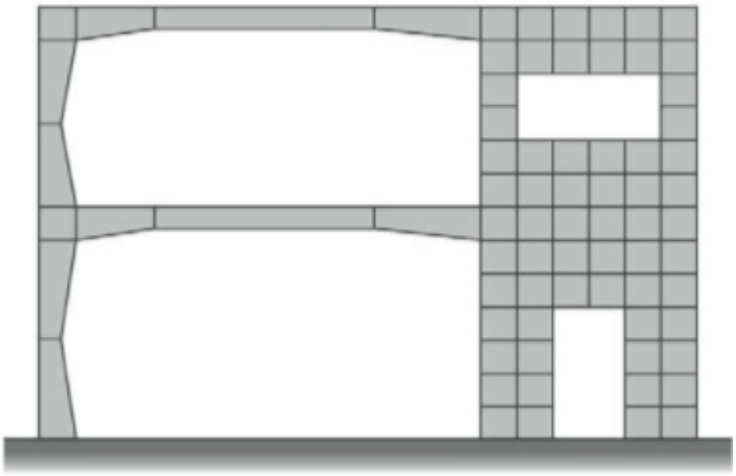
1. Facilidade na descrição da geometria da estrutura;
2. Habilidade em considerar os materiais compósitos e anisotrópicos;

3. Habilidade no tratamento das descontinuidades;
4. Facilidade no tratamento das condições de carga combinadas;
5. Habilidade no tratamento dos carregamentos térmicos e dinâmicos;
6. Habilidade no tratamento de problemas estruturais não-lineares;
7. Capacidade para uma completa automatização dos cálculos.

Para análise de vigas, o tipo de elemento adotado são os elementos de barras. A rede de elementos e nós de uma região é denominada malha, sendo ela mais densa conforme mais elementos são utilizados. Para resultados mais precisos, aumentar a densidade da malha pode ser necessária.

A figura 5 demonstra uma representação esquemática de uma estrutura discretizada por elementos finitos, onde a estrutura é composta de elementos de seção variável, tema abordado neste trabalho.

Figura 5 – Representação esquemática de um prédio com vigas, colunas e paredes



Fonte: Ugural (2009, p. 571)

4.4 Método dos Elementos Finitos aplicado a vigas retangulares de altura variável

Para analisar o equilíbrio de um elemento finito de viga, existem alguns métodos que podem ser utilizados, mas o Princípio dos Trabalhos Virtuais, ou PTV se destaca entre eles. O PTV, estabelece que um sistema mecânico estará em equilíbrio caso o trabalho virtual de todas as forças atuantes no elemento for nulo. Assim, para qualquer deslocamento virtual imposto ao elemento, pode-se definir que o trabalho

virtual externo produzido pelas forças externas quando ocorrem deslocamentos virtuais é igual ao trabalho virtual interno produzido pelos esforços reais internos quando ocorrem as deformações virtuais como está descrito na equação (5).

$$U_e^* = U_i^* \tag{5}$$

Para a parcela do trabalho virtual externo, escreve-se como:

$$U_e^* = F.\Delta^* + M\phi^* + \int_0^L q(x).\Delta(\mathrm{x})^* \, dx \tag{6}$$

Segundo Han *et al.* (1999), admitindo a teoria de Euler-Bernoulli, a parcela da energia das tensões de cisalhamento é desprezada, assim, escreve-se:

$$U_i^* = \int_V \sigma \, . \delta\varepsilon^* \, dV \tag{7}$$

As deformações em um elemento fletido podem ser determinadas a partir a relação diferencial entre deformação-deslocamento (BEER; JOHNSTON; MAZUREK, 2012):

$$\varepsilon = y.\frac{d^2v}{dx^2} \tag{8}$$

Considerando-se o comportamento elástico-linear do material e a lei de Hooke, tem-se:

$$\sigma = E.\varepsilon \tag{9}$$

Substituindo (8) em (9), obtém-se:

$$\sigma = E\left(y.\frac{d^2v}{dx^2}\right) \tag{10}$$

Então, substituindo (8) e (10) em (7), temos:

$$U_i^* = \int_V E\left(y.\frac{d^2v}{dx^2}\right).\left(y.\frac{d^2\delta v^*}{dx^2}\right) dV \tag{11}$$

A integral de volume apresentada na equação (11) pode ser transformada em uma integral dupla na área e no comprimento do elemento, como segue:

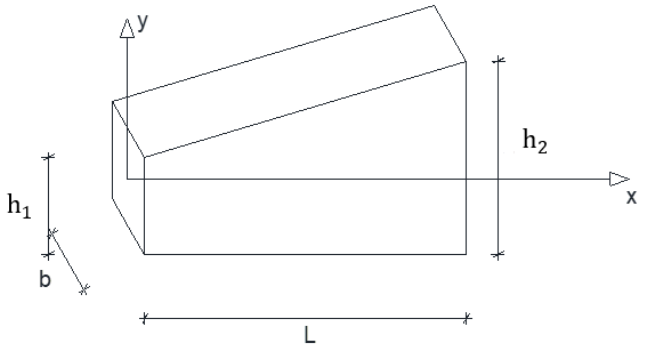
$$U_i^* = E \int_0^L \left(\frac{d^2 v}{dx^2}\right) . \left(\frac{d^2 \delta v^*}{dx^2}\right) . \left(\int_A y^2 dA\right) dx \tag{12}$$

A integral da área contida na equação (12) define o momento de inércia da seção transversal em relação ao seu eixo neutro (BITTENCOURT E FEIJÓO, 1999).

Neste trabalho, considera-se que a viga possui seção transversal retangular com altura variando linearmente ao longo do comprimento do elemento estrutural, conforme ilustrado na figura 6. Desta forma, a altura da seção transversal em função do comprimento do elemento pode ser descrita por meio da equação (13).

$$h(x) = \left(1 - \frac{x}{L}\right) h_1 + \left(\frac{x}{L}\right) h_2 \tag{13}$$

Figura 6 – Modelo da variação de altura do elemento finito.



Fonte: Autoria própria (2021)

O momento de inércia de uma seção retangular em relação ao seu eixo centroidal pode ser calculado através da equação que segue:

$$I = \frac{b.h^3}{12} \tag{14}$$

Substituindo (13) em (14), tem-se:

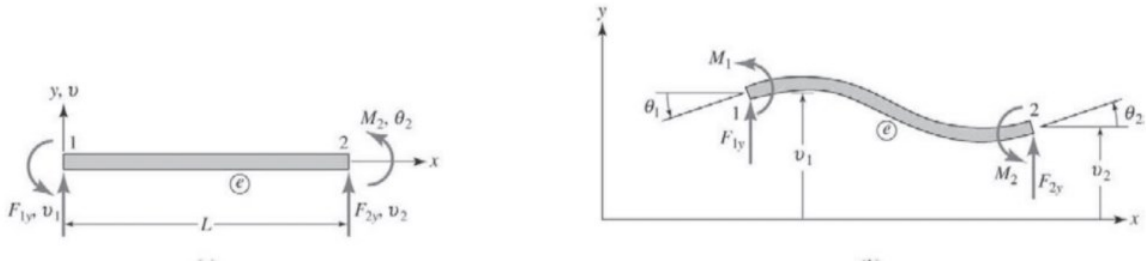
$$I(x) = \frac{b}{12}\left[\left(1 - \frac{x}{L}\right) h_1 + \left(\frac{x}{L}\right) h_2\right]^3 \tag{15}$$

Substituindo (15) em (12), obtém-se:

$$U_i^* = E\int_0^L \left(\frac{d^2v}{dx^2}\right).\left(\frac{d^2\delta v^*}{dx^2}\right).\left\{\frac{b}{12}\left[\left(1-\frac{x}{L}\right)h_1+\left(\frac{x}{L}\right)h_2\right]^3\right\}dx \qquad (16)$$

A figura 7 demonstra as variáveis nodais que serão utilizadas para o desenvolvimento da matriz de rigidez para o elemento finito de viga. No elemento são considerados forças nodais F, contido no plano do elemento e perpendicular ao seu eixo longitudinal, e momentos nodais M, no plano da seção transversal do elemento; demonstra-se também os deslocamentos nodais v e os giros nodais de seção transversal $\theta = \frac{dv}{dx}$.

Figura 7 – Variáveis nodais



Fonte: Ugural (2009, p. 572)

Segundo Ugural (2009), em um elemento finito de viga de rigidez constante onde se admite que não há carregamento entre os nós, ou seja, apenas forças e momentos nodais, o deslocamento vertical dos pontos pertencentes ao seu eixo médio pode ser assumido como uma função polinomial cúbica na coordenada x.

A variação de rigidez do elemento introduzida neste trabalho estabelece soluções analíticas para deslocamento mais complexas por envolver integrais de divisão de polinômios. Nesse sentido, neste trabalho será utilizada a mesma função polinomial cúbica já referida para interpolar os deslocamentos ao longo do domínio do elemento. A precisão necessária para os resultados deverá ser verificada a partir de análises de refinamento de malha. Assim, tem-se:

$$v(x) = ax^3 + bx^2 + cx + d \qquad (17)$$

Então a inclinação θ:

$$\theta = \frac{dv}{dx} = 3ax^2 + 2bx + c \tag{18}$$

Assumindo-se então os deslocamentos e giros nodais condições de contorno, define-se as constantes presentes na equação (17), como segue:

$$v(x) = x^3\left(\frac{\theta_1}{L^2} + \frac{\theta_2}{L^2} + \frac{2v_1}{L^3} + \frac{2v_2}{L^3}\right) + x^2\left(-\frac{2\theta_1}{L} - \frac{\theta_2}{L} - \frac{3v_1}{L^2} - \frac{3v_2}{L^2}\right) + x\theta_1 + v_1 \tag{19}$$

Reordenando a equação (19) em função dos deslocamentos e giros nodais, tem-se:

$$v(x) = \left(1 - \frac{3x^2}{L^2} + \frac{2x^3}{L^3}\right)v_1 + \left(x - \frac{2x^2}{L} + \frac{x^3}{L^2}\right)\theta_1 + \left(\frac{3x^2}{L^2} - \frac{2x^3}{L^3}\right)v_2 + \left(-\frac{x^2}{L} + \frac{x^3}{L^2}\right)\theta_2 \tag{20}$$

E simplificando:

$$v(x) = \phi_1 v_1 + \phi_2\theta_1 + \phi_3 v_2 + \phi_4\theta_2 \tag{21}$$

sendo:

$$\phi_1 = \left(1 - \frac{3x^2}{L^2} + \frac{2x^3}{L^3}\right)$$
$$\phi_2 = \left(x - \frac{2x^2}{L} + \frac{x^3}{L^2}\right)$$
$$\phi_3 = \left(\frac{3x^2}{L^2} - \frac{2x^3}{L^3}\right)$$
$$\phi_4 = \left(-\frac{x^2}{L} + \frac{x^3}{L^2}\right)$$

Substituindo (21) em (18), e adotando-se a mesma função de aproximação para representar o deslocamento virtual ao longo do elemento, tem-se:

$$\begin{aligned} U_i^* = E\int_0^L \left(\frac{d^2\phi_1}{dx^2}\right)\left(\frac{d^2\phi_1}{dx^2}\right)\left(\frac{b}{12}\left[\left(1 - \frac{x}{L}\right)h_1 + \left(\frac{x}{L}\right)h_2\right]\right) v_1.\delta v_1^* + \\ + \int_0^L \left(\frac{d^2\phi_1}{dx^2}\right)\left(\frac{d^2\phi_2}{dx^2}\right)\left(\frac{b}{12}\left[\left(1 - \frac{x}{L}\right)h_1 + \left(\frac{x}{L}\right)h_2\right]\right) v_1.\delta\theta_1^* + \cdots + \\ + \int_0^L \left(\frac{d^2\phi_4}{dx^2}\right)\left(\frac{d^2\phi_4}{dx^2}\right)\left(\frac{b}{12}\left[\left(1 - \frac{x}{L}\right)h_1 + \left(\frac{x}{L}\right)h_2\right]\right) \theta_2.\delta\theta_2^* \end{aligned} \tag{22}$$

Ou, para rearranjar de maneira matricial:

$$\varphi_{ij} = \int_0^L \left(\frac{d^2\phi_i}{dx^2}\right)\left(\frac{d^2\phi_j}{dx^2}\right)\left\{\frac{b}{12}\left[\left(1-\frac{x}{L}\right)h_1 + \left(\frac{x}{L}\right)h_2\right]^3\right\}dx \tag{23}$$

$$U_i^* = E\begin{bmatrix}\varphi_{11} & \varphi_{12} & \varphi_{13} & \varphi_{14}\\ \varphi_{21} & \varphi_{22} & \varphi_{23} & \varphi_{24}\\ \varphi_{31} & \varphi_{32} & \varphi_{33} & \varphi_{34}\\ \varphi_{41} & \varphi_{42} & \varphi_{43} & \varphi_{44}\end{bmatrix}\begin{bmatrix}\upsilon_1\\ \theta_1\\ \upsilon_2\\ \theta_2\end{bmatrix}\begin{bmatrix}\delta\upsilon_1^*\\ \delta\theta_1^*\\ \delta\upsilon_2^*\\ \delta\theta_2^*\end{bmatrix}^T \tag{24}$$

Considerando que os deslocamentos virtuais são unitários e inserindo a parcela do trabalho externo, obtém-se o sistema de equações algébricas para um elemento finito, como segue:

$$\frac{E\cdot b}{60\cdot L^3}\begin{bmatrix}k_{11} & k_{12} & k_{13} & k_{14}\\ & k_{22} & k_{23} & k_{24}\\ & & k_{33} & k_{34}\\ Sim. & & & k_{44}\end{bmatrix}\begin{bmatrix}\upsilon_1\\ \theta_1\\ \upsilon_2\\ \theta_2\end{bmatrix} = \begin{bmatrix}F_1\\ M_1\\ F_2\\ M_2\end{bmatrix} \tag{25}$$

sendo:

$$k_{11} = 21{h_1}^3 + 9{h_1}^2h_2 + 9h_1{h_2}^2 + 21{h_2}^3$$

$$k_{12} = L\left(15{h_1}^3 + 6{h_1}^2h_2 + 3h_1{h_2}^2 + 6{h_2}^3\right)$$

$$k_{13} = -21{h_1}^3 - 9{h_1}^2h_2 - 9h_1{h_2}^2 - 21{h_2}^3$$

$$k_{14} = L\left(6{h_1}^3 + 3{h_1}^2h_2 + 6h_1{h_2}^2 + 15{h_2}^3\right)$$

$$k_{22} = L^2\left(11{h_1}^3 + 5{h_1}^2h_2 + 2h_1{h_2}^2 + 2{h_2}^3\right)$$

$$k_{23} = L\left(-15{h_1}^3 - 6{h_1}^2h_2 - 3h_1{h_2}^2 - 6{h_2}^3\right)$$

$$k_{24} = L^2\left(4{h_1}^3 + {h_1}^2h_2 + h_1{h_2}^2 + 4{h_2}^3\right)$$

$$k_{33} = 21{h_1}^3 + 9{h_1}^2h_2 + 9h_1{h_2}^2 + 21{h_2}^3$$

$$k_{34} = L\left(-6{h_1}^3 - 3{h_1}^2h_2 - 6h_1{h_2}^2 - 15{h_2}^3\right)$$

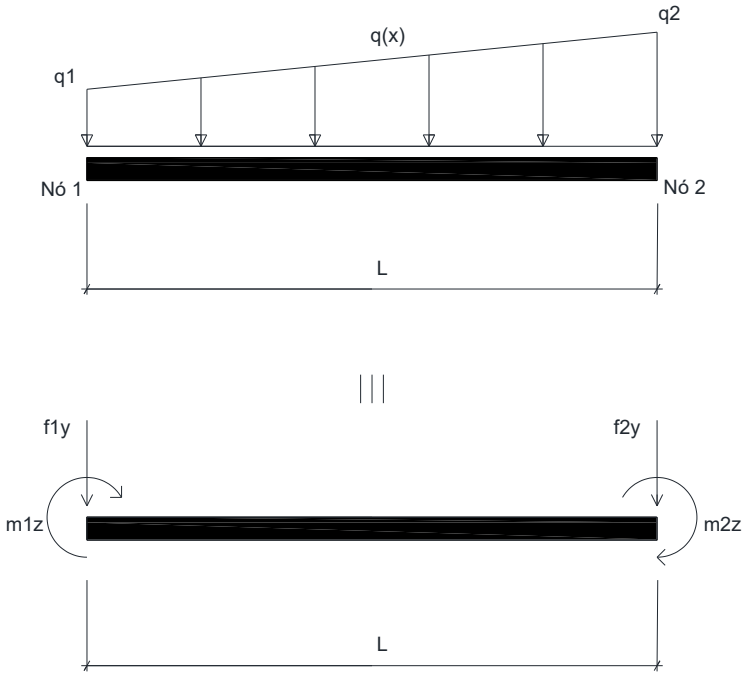
$$k_{44} = L^2\left(2{h_1}^3 + 2{h_1}^2h_2 + 5h_1{h_2}^2 + 11{h_2}^3\right)$$

Ou, de maneira simplificada:

$$[k_e]\{\delta_e\} = \{F_e\} \tag{26}$$

Na equação (26) o vetor de forças $\{F_e\}$ a ser considerado será uma associação das forças externas concentradas e dos carregamentos distribuídos sob o elemento concentrados na forma de forças nodais equivalentes. A figura 8 ilustra o sistema de forças nodais equivalentes para carregamentos distribuídos linearmente no elemento.

Figura 8 - Esforços equivalentes ao carregamento distribuído



Fonte: Autoria própria (2021)

Assim, para determinar a equação de q(x), assumindo-se um carregamento linear, tem-se:

$$q(x) = a.x + b \tag{27}$$

Definindo, então, as condições de contorno $q(x = 0) = q_1$ e $q(x = L) = q_2$, tem-se:

$$q(x) = \frac{q_2 - q_1}{L}.x + q_1 \tag{28}$$

Na equação (6), trabalhando apenas com a parcela de energia referente ao carregamento distribuído e considerando que os deslocamentos verticais são descritos pela equação (16) e o carregamento linear é descrito pela equação (25), tem-se:

$$\int_0^L q(x).\Delta(\mathrm{x})^* \, dx = \int_0^L \left(\frac{q_2-q_1}{L}.x + q_1\right).\left(\varphi_1 v_{1y} + \varphi_2 \phi_{1z} + \varphi_3 v_{2y} + \varphi_4 \phi_{2z}\right) dx \quad (29)$$

Assim, resolvendo a equação (28) e considerando-se os deslocamentos virtuais unitários, obtém-se a parcela dos esforços equivalentes para cada nó do elemento em análise:

$$\{f\} = \begin{bmatrix} f_{1y} \\ m_{1z} \\ f_{2y} \\ m_{2z} \end{bmatrix} = \begin{bmatrix} \frac{7L}{20}.q_1 + \frac{3L}{20}.q_2 \\ \frac{L^2}{20}.q_1 + \frac{L^2}{30}.q_2 \\ \frac{3L}{20}.q_1 + \frac{7L}{20}.q_2 \\ -\left(\frac{L^2}{30}.q_1 + \frac{L^2}{20}.q_2\right) \end{bmatrix} \quad (30)$$

4.5 Linguagem Python

A linguagem Python é uma linguagem de programação de alto nível lançada por Guido van Rossum em 1991, e possui um modelo de desenvolvimento aberto gerenciado pela organização Python Software Foundation.

Segundo Yuill e Halpin (2006), Python é apenas uma linguagem de programação de muitas, assim como a linguagem da humanidade, existem muitas linguagens de programação, como JAVA, LISP, PHP e Pearl, e cada linguagem é boa em pelo menos uma coisa, como Java para softwares para aparelhos móveis, ou PHP para bancos de dados. Mas no geral, todas essas linguagens possuem o mesmo modelo – A maioria tem dados em variáveis e funções para que sejam trabalhados. Python é uma linguagem de programação que é fácil de ler e se entender, é gratuita e possui uma grande e amigável comunidade.

Uma das diversas funções do Python é a possibilidade de importar bibliotecas, uma coleção de módulos de script com a finalidade de simplificar os processos de programação e remover a necessidade de reescrever comandos mais usados.

Dentre as bibliotecas, as que mais chama atenção ao trabalho desenvolvido é a biblioteca numpy, biblioteca que oferece funções matemáticas de álgebra linear, com a capacidade de resolver matrizes quadradas com simples chamadas de função.

Também entre as bibliotecas há a biblioteca matplotlib.pyplot, que possui funções que quando chamadas automatizam a geração de gráficos pelo Python.

5 ASPECTOS COMPUTACIONAIS

Para a implementação do código computacional optou-se pelo uso da linguagem Python, utilizando-se as bibliotecas *numpy* e *matplotlib.pyplot*, como ferramentas para resolução de sistemas de equações e geração automática de gráficos.

O código foi desenvolvido com diversas funções agrupadas em três grupos, sendo eles: organização de dados, processamento e saída de dados.

No grupo de organização de dados está prevista a leitura do arquivo de dados (“entrada.txt”), onde o usuário insere os parâmetros do problema, seguindo a sequência pré-determinada, de acordo com a estrutura elaborada demonstrada no quadro 1:

Quadro 1 – Modelo do arquivo de entrada

Número de nós	Número de elementos	Coeficiente de detalhamento				
número do nó 1	coordenada do nó 1	restrição y no nó 1	restrição z no nó 1	força concentrada y no nó 1	momento z concentrado no nó 1	altura do nó 1
número do nó 2	coordenada do nó 2	restrição y no nó 2	restrição z no nó 2	força concentrada y no nó 2	momento z concentrado no nó 2	altura do nó 2
número do nó n	coordenada do nó n	restrição y no nó n	restrição z no nó n	força concentrada y no nó n	momento z concentrado no nó n	altura do nó n
número do elemento 1	nó inicial do elemento	nó final do elemento	carga distribuída inicial do elemento 1	carga distribuída final do elemento 1	módulo de elasticidade do elemento 1	base do elemento 1
número do elemento 2	nó inicial do elemento	nó final do elemento	carga distribuída inicial do elemento 2	carga distribuída final do elemento 2	módulo de elasticidade do elemento 2	base do elemento 2
número do elemento n	nó inicial do elemento	nó final do elemento	carga distribuída inicial do elemento n	carga distribuída final do elemento n	módulo de elasticidade do elemento n	base do elemento n

Fonte: Autoria própria (2021)

Apresenta-se na sequência o detalhamento das informações sintetizadas no quadro 1:

- Número de nós: quantidade de nós que apresentam restrição de deslocamento e/ou giro ou ação de força e/ou momento concentrado.

Deve-se incluir também neste total os nós da extremidade da viga, independente da presença de vínculos ou carregamentos;

- Número de elementos: quantidade total de elementos, sendo estes as barras entre os nós descritos;
- Coeficiente de detalhamento: valor inserido para que o código aumente automaticamente o número de nós e elementos finitos a fim de se realizar simulações numéricas com malhas mais refinadas. Por exemplo, caso se insira o valor 0,1 o software irá inserir nós a cada 0,1 unidades dimensionais e elementos entre eles;
- Coordenada do nó “n”: valor inserido da coordenada do nó descrito, sendo, necessariamente, a coordenada do primeiro nó igual a 0 e a coordenada do enésimo nó igual ao comprimento total da viga em análise;
- Restrição y no nó “n”: caso haja uma restrição de deslocamento em y no nó (apoio ou engaste), insere-se o valor 1; caso contrário, insere-se o valor 0;
- Restrição z no nó “n”: caso haja uma restrição de rotação e torno do eixo z no nó (engaste), insere-se o valor 1; caso contrário, insere-se o valor 0;
- Força concentrada y no nó “n”: valor inserido representando a presença de uma força concentrada no nó com orientação definida pelo sistema de eixos cartesianos;
- Momento z concentrado no nó “n”: valor inserido representando a presença de um momento fletor concentrado no nó, sendo sentido horário negativo, e anti-horário positivo;
- Altura no nó “n”: valor da altura da seção transversal no nó em questão;
- Nó inicial do elemento “n”: número do nó onde se inicia o elemento em questão;
- Nó final do elemento “n”: número do nó onde se encerra o elemento em questão;
- Carga distribuída inicial do elemento “n”: valor do carregamento distribuído no nó inicial do elemento em questão, conforme explicitado na figura 8;
- Carga distribuída final do elemento “n”: valor do carregamento distribuído no nó final do elemento em questão, conforme explicitado na figura 8;

- Módulo de elasticidade do elemento “n”: valor do módulo de elasticidade E do elemento em questão;
- Base do elemento “n”: largura da seção transversal do elemento em questão.

De forma a melhor visualizar como deve ser realizada a entrada de dados, no apêndice B é apresentado o arquivo de entrada de dados referente ao exemplo 3 analisado no item 6 deste trabalho.

Além da leitura de dados, também estão presentes no grupo organização de dados as funções abertura e detalhador, cujas finalidades são descritas nos subitens que seguem.

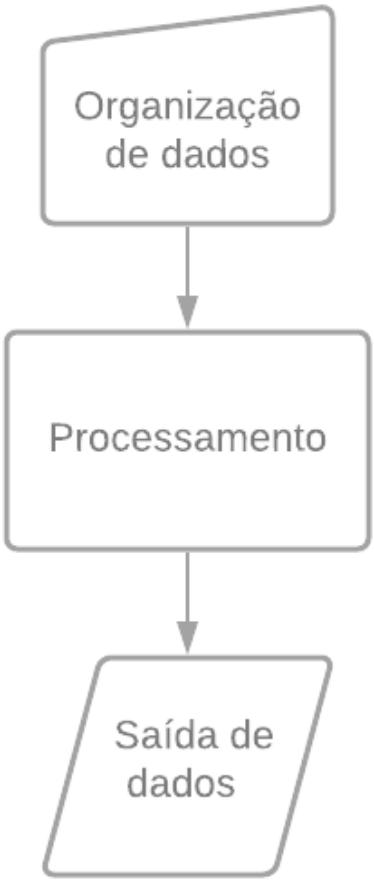
No grupo de processamento estão as funções que realizam todos os cálculos necessários, a partir dos dados previamente organizados pelo grupo anterior, para que se obtenha os deslocamentos e esforços nodais desejados na análise estrutural em questão. Dentro deste grupo estão presentes funções para elaboração do vetor de forças nodais equivalentes, elaboração do vetor de cargas, montagem da matriz de rigidez do elemento, montagem da matriz de rigidez da estrutura, a resolução do sistema de equações, cálculo do momento fletor e da força cortante em cada nó e, por fim, o cálculo das reações nos apoios.

No grupo de saída de dados está presente a função que recebe os deslocamentos, o momento fletor e a força cortante em cada nó e exporta, em formato (“jpeg”), a estrutura deformada e os diagramas de momento fletor e força cortante.

5.1 Estrutura do código

O fluxograma da figura 9 demonstra de maneira gráfica a organização da estrutura do código computacional, contendo os grupos de funções utilizadas e exibindo o fluxo de informações ao longo código.

Figura 9 – Fluxograma do código desenvolvido



Fonte: Autoria própria (2021)

5.2 Grupos de funções

5.2.1 Organização de dados

5.2.1.1 Função Abertura

Função onde é efetuada a leitura dos dados do arquivo “entrada.txt”, previamente fornecido pelo usuário, abrindo-o e alocando os valores em matrizes. Após isso ocorre o fechamento do arquivo.

5.2.1.2 Função detalhador

Função que divide a estrutura alocando um nó a cada “n” unidades de medida, conforme parâmetro fornecido pelo usuário no arquivo de entrada de dados. São também inseridos elementos entre os nós alocados.

Esta função tem como objetivo gerar automaticamente a malha de elementos finitos possibilitando, assim, análises numéricas utilizando malhas mais refinadas, bem como fornecer diagramas de esforços solicitantes mais detalhados. Assim, caso seja utilizado, cabe ao usuário somente especificar os nós onde há vínculos, forças e/ou momentos concentrados, início e fim de forças distribuídas, mudança de altura e/ou largura de seção transversal e mudança de módulo de elasticidade, caso a viga seja composta por mais de um material.

Nesta função, são também calculadas as alturas inicial e final dos elementos finitos que foram gerados automaticamente.

5.2.2 Processamento

5.2.2.1 Função processarmatlocal

Cria a partir da equação (25) a matriz de rigidez local dos elementos e as fornece à função matglobal para a montagem da matriz global da estrutura.

5.2.2.2 Função matglobal

Aloca as matrizes de rigidez local dos elementos na matriz de rigidez global da estrutura, sendo esta uma matriz quadrada de dimensões dadas pelo produto do número de graus de liberdade de cada nó (dois) pelo número total de nós da estrutura.

A incidência de cada matriz local na matriz global é definida em função do grau de liberdade e do nó a que se refere. Assim, para nós comuns a dois elementos, há uma sobreposição das matrizes locais na matriz global.

5.2.2.3 Função vetforças

Cria o vetor de forças global adicionando às forças e aos momentos concentrados fornecidos pelo usuário as forças nodais equivalentes calculadas, por meio da equação (30), nos elementos que possuem carregamento distribuídos em sua extensão.

5.2.2.4 Função contorno

Recebe a matriz de rigidez e o vetor de forças global e, a partir das informações dos nós vinculados descritas no arquivo de entrada de dados, impõe, no sistema de equações lineares, a condição de deslocamento nodal e/ou giro nodal nulo.

O procedimento é definido a partir da atribuição do valor unitário na diagonal principal da linha da matriz de rigidez global correspondente ao grau de liberdade restrito e atribuindo zero a todos os outros valores da linha e da coluna. Além disso, impõe-se zero no vetor de cargas global na posição referente ao grau de liberdade restrito. Esse procedimento, de maneira genérica, é melhor visualizado na equação (31).

$$\begin{bmatrix} K_{11} & 0 & K_{13} & K_{14} & 0 & K_{16} & \cdots & K_{1n} \\ 0 & 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ K_{31} & 0 & K_{33} & K_{34} & 0 & K_{36} & \dots & K_{3n} \\ K_{41} & 0 & K_{43} & K_{44} & 0 & K_{46} & \dots & K_{4n} \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ K_{61} & 0 & K_{63} & K_{64} & 0 & K_{66} & \dots & K_{6n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{n1} & 0 & K_{n3} & K_{n4} & 0 & K_{n6} & \dots & K_{nn} \end{bmatrix} * \begin{Bmatrix} \upsilon_1 \\ \theta_1 \\ \upsilon_1 \\ \theta_2 \\ \upsilon_3 \\ \theta_3 \\ \vdots \\ \theta_n \end{Bmatrix} = \begin{Bmatrix} F_{1y} \\ 0 \\ F_{2y} \\ M_{2z} \\ 0 \\ M_{3z} \\ \vdots \\ M_{nz} \end{Bmatrix} \quad (31)$$

5.2.2.5 Função solução

Utiliza-se a biblioteca numpy para resolver o sistema linear de equações $K \cdot u = F$ e obtém-se os deslocamentos e giros de cada nó.

5.2.2.6 Função retorno

A partir dos valores dos deslocamentos e giros obtidos para cada nó, realiza o produto entre a matriz de rigidez, sem condições de contorno, e o vetor de deslocamentos, obtendo-se, assim, os vetores de forças e momentos nodais.

5.2.2.7 Função distribuída

Subtrai-se do vetor obtido na função retorno os valores das forças distribuídas e das forças e momentos concentrados, restando somente as reações de apoio

5.2.2.8 Função esforçosinternos

Utiliza-se a função processarmatlocal para realizar o produto demonstrado pela equação (25) para cada elemento, obtendo-se o valor da força cortante e momento fletor no nó inicial e no nó final do elemento. A função ainda os organiza e fornece os dados para a função plotagem.

5.2.3 Saída de dados

5.2.3.1 Função plotagem

Utiliza a biblioteca *matplotlib.pyplot* para plotar a configuração deformada da estrutura e os diagramas de força cortante e momento fletor do elemento estrutural, demonstrando seus pontos de máximo e mínimo, sendo salvos no formato (“jpeg”) no pasta local do software.

6 RESULTADOS E DISCUSSÕES

Visando validar a formulação desenvolvida e o software implementado neste trabalho, alguns exemplos de vigas são apresentados neste item. Os exemplos estudados foram apresentados por outros pesquisadores ou então idealizados pelo próprio autor. Neste último caso, os resultados fornecidos pelo software implementado foram comparados com análises realizadas no software comercial SCIA Engineer 20.0 (versão educacional).

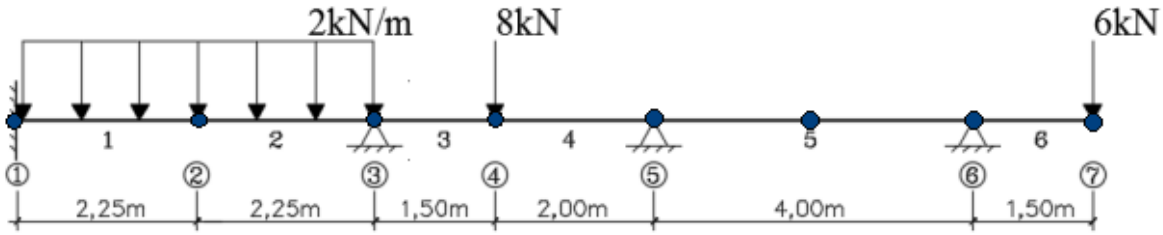
6.1 Exemplo 1

Neste exemplo serão realizadas comparações entre os resultados referentes aos deslocamentos e giros de seção transversal e as forças internas obtidas por Rodrigues (2016).

O problema abordado consiste em uma viga contínua vinculada ao meio externo através de engaste e apoios conforme ilustra a figura 10. Na figura, também é apresentado o carregamento atuante no elemento, sendo este composto por duas forças concentradas e um carregamento uniformemente distribuído em seu primeiro tramo.

A viga possui seção transversal constante com largura de 15 cm e altura de 30 cm e é constituída por um material elástico com módulo de elasticidade $E = 3{,}0 \text{ x } 10^7 \ kN/m^2$. Em sua análise, Rodrigues (2016) utilizou uma malha composta por 6 elementos finitos e 7 nós, como indica a figura 10.

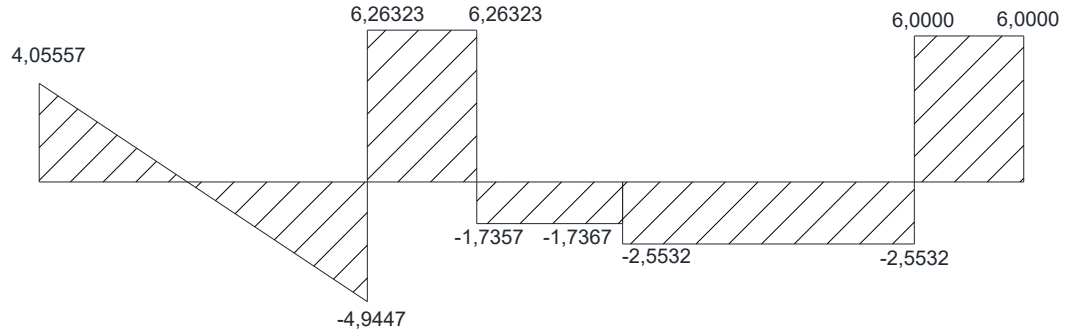
Figura 10 – Viga submetida a carregamentos variados.



Fonte: Adaptado de Rodrigues (2016, p. 52)

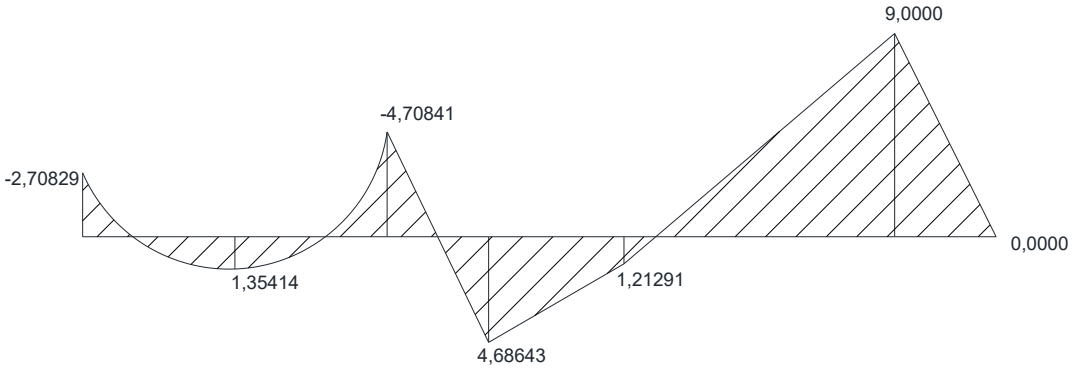
Rodrigues (2016) obteve como resultados os diagramas de força cortante e de momento fletor ilustrados nas figuras 11 e 12, respectivamente.

Figura 11 – Diagrama de força cortante para o exemplo 1, em kN



Fonte: Adaptado de Rodrigues (2016, p. 52)

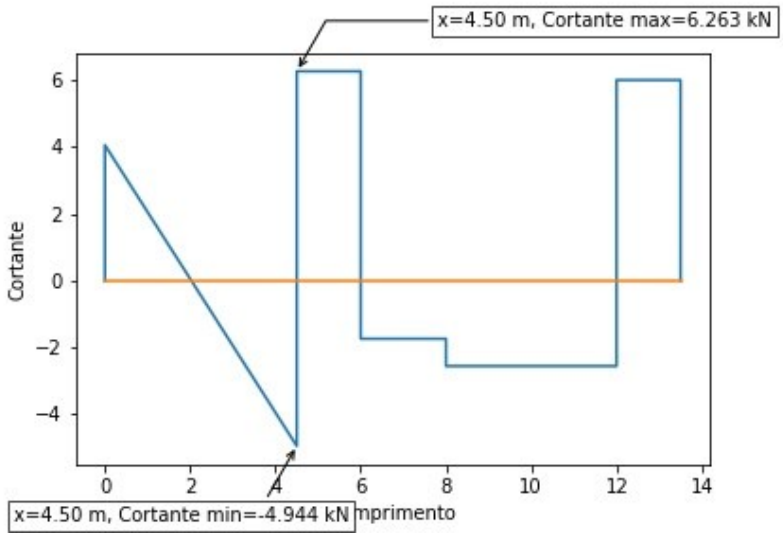
Figura 12 – Diagrama de momento fletor para o exemplo 1, em kN.m



Fonte: Adaptado de Rodrigues (2016, p. 53)

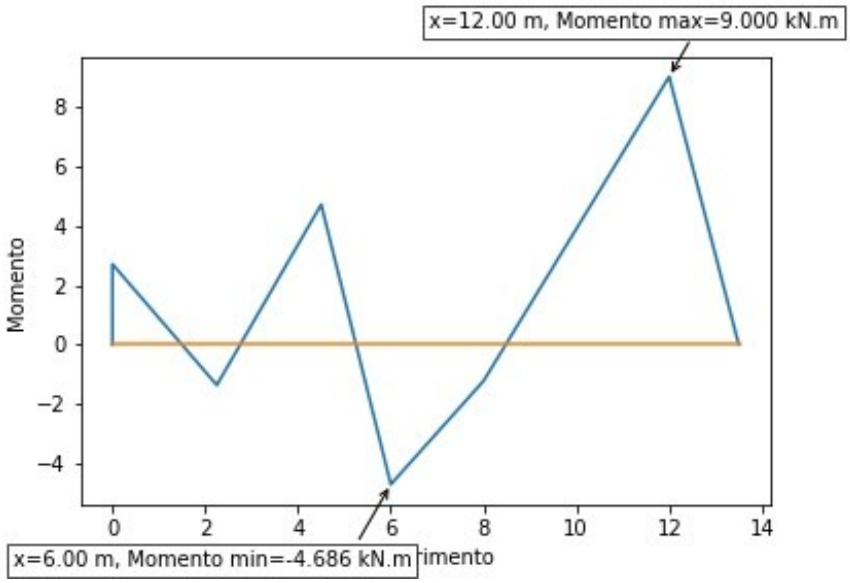
Utilizando-se da mesma malha fornecida pela autora, a análise foi realizada utilizando-se o código implementado neste estudo. Os diagramas de força cortante e momento fletor gerados pelo próprio software são ilustrados nas figuras 13 e 14, respectivamente. Vale mencionar que o próprio software indica, em destaque, a posição e os valores de esforços máximo e mínimo ao longo do elemento.

Figura 13 – Diagrama de força cortante obtido pelo software.



Fonte: Autoria própria (2021)

Figura 14 – Esboço de diagrama de momento fletor obtido pelo software.



Fonte: Autoria própria (2021)

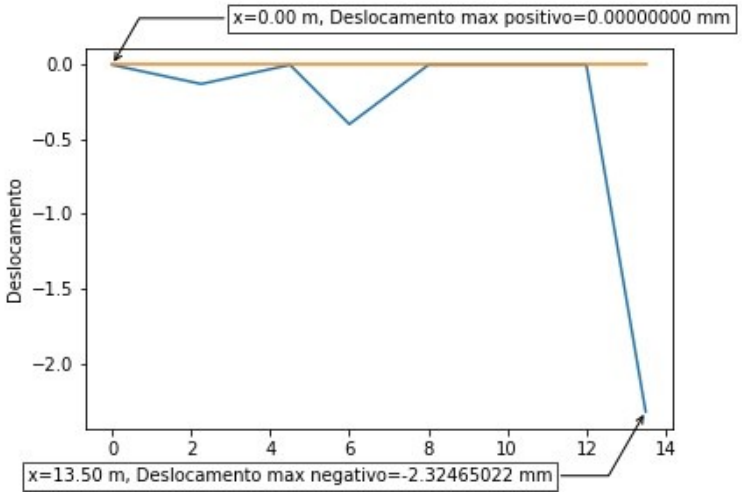
Comparando-se os diagramas de força cortante (figuras 11 e 13) verifica-se que os resultados nodais obtidos pelo software implementado são os mesmos fornecidos por Rodrigues (2016). Além disso, como o código implementado prevê, na construção dos diagramas, a ligação dos valores nodais a partir de segmentos de reta, o diagrama de força cortante apresenta-se com o traçado correto.

Analisando-se comparativamente os diagramas de momento fletor (figuras 12 e 14) pode-se chegar a mesma conclusão em termos de valores nodais. No entanto, a forma como foi programada computacionalmente, o traçado dos diagramas não

fornece, neste caso, o traçado correto para o Momento Fletor, principalmente no que diz respeito ao tramo da viga em que há a presença do carregamento uniformemente distribuído. Assim, para que se tenha um traçado mais adequado do diagrama neste trabalho é necessário que se realize a análise utilizando uma malha mais refinada, inserindo-se um “coeficiente de detalhamento” menor no arquivo de entrada de dados.

Por fim, na figura 15 é ilustrada um esboço da configuração deformada do elemento estrutural fornecida pelo software implementado a partir da análise utilizando a malha de elementos finitos anteriormente descrita.

Figura 15 – Esboço da configuração deformada do elemento estrutural.

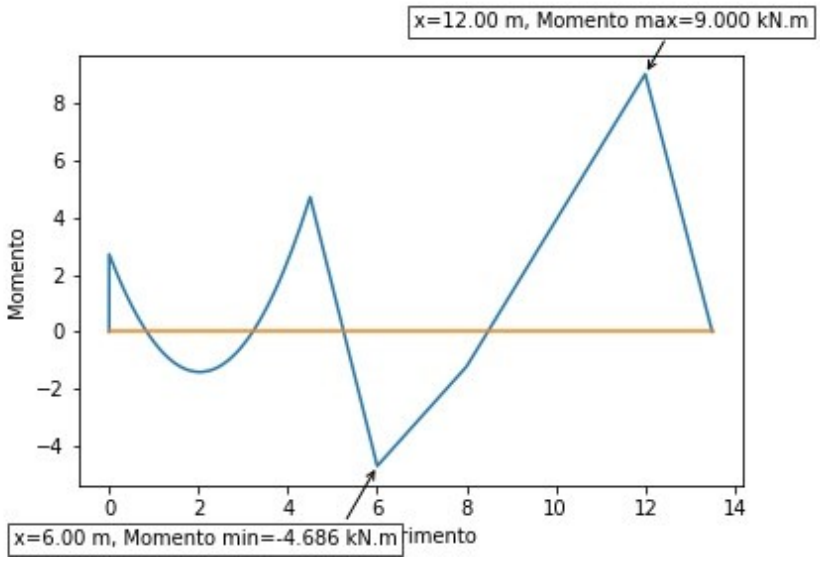


Fonte: Autoria própria (2021)

Em se tratando da configuração deformada do elemento estrutural (figura 15), pode-se chegar à mesma conclusão: neste trabalho, o traçado mais adequado da curva só é possível utilizando-se malhas mais refinadas, visto que não é realizado pós processamento pelo código.

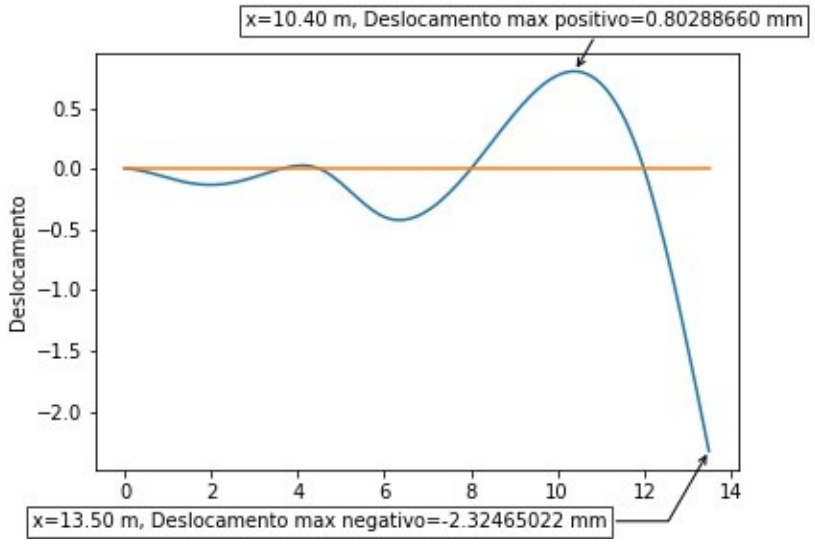
Visando, então, a construção do diagrama de momento fletor e da configuração deformada mais adequados, uma nova análise foi realizada utilizando-se um “coeficiente de detalhamento” igual a 0,1 m. Os resultados obtidos são apresentados nas figuras 16 e 17.

Figura 16 – Diagrama de momento fletor com malha refinada.



Fonte: Autoria própria (2021)

Figura 17 – Configuração deformada do elemento estrutural com malha refinada.



Fonte: Autoria própria (2021)

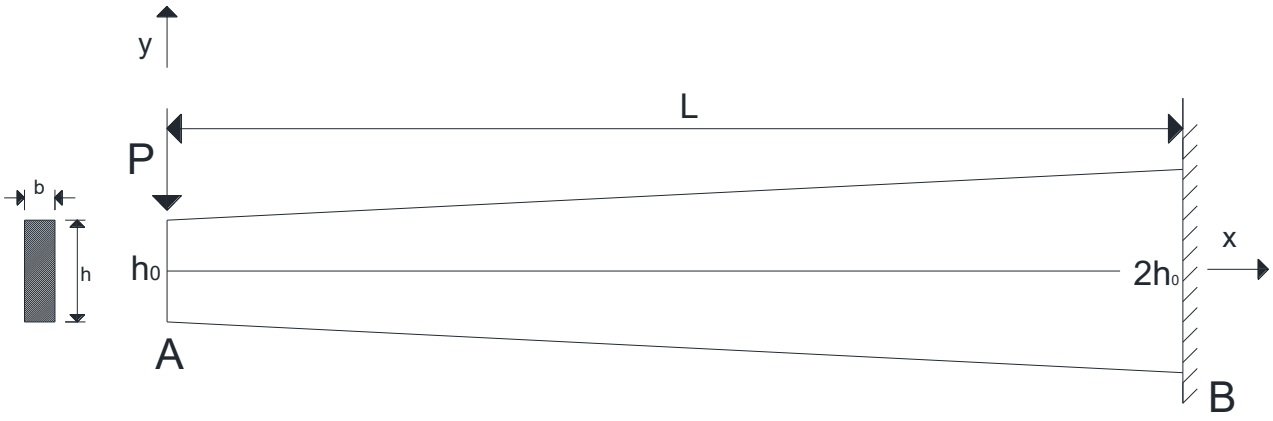
As figuras 16 e 17 ilustram os traçados mais adequados do diagrama de momento fletor e da configuração deformada da estrutura. Percebe-se claramente que, na análise inicial, alguns detalhes das curvas haviam sido perdidos, principalmente no trecho da viga que apresenta deslocamentos positivos.

Por fim, apresentando valores nodais exatamente iguais aos obtidos por Rodrigues (2016) e curvas com traçados adequados, pode-se garantir a eficiência da formulação desenvolvida para casos em que há a altura constante e em estruturas hiperestáticas.

6.2 Exemplo 2

Neste exemplo, proposto por Ugural (2009), é simulada uma viga em balanço de comprimento $L = 5\ m$ sujeita a ação de uma força concentrada $P = -10\ kN$ em sua extremidade livre. A viga possui seção transversal com largura $b = 15\ cm$ e altura variando linearmente ao longo de seu comprimento, tendo valor inicial $h_0 = 50\ cm$ e valor final igual a $2h_0$. Ainda, assume-se que o material que a compõe possui módulo de elasticidade longitudinal $E = 30\ GPa.$ A figura 18 ilustra o elemento estrutural em questão.

Figura 18 – Viga de altura variável em balanço submetida a carga concentrada.



Fonte: Autoria própria (2021)

Para o problema em questão, pode-se assumir a lei de variação de altura ao longo do comprimento conforme apresentado na equação (32).

$$h(x) = h_0\left(1+\frac{x}{L}\right) = \frac{h_0}{L}(L+x) \tag{32}$$

Substituindo-se a equação (33) na expressão de momento de inércia de uma seção transversal retangular em relação ao seu eixo centroidal, obtém-se:

$$I = \frac{I_A}{L^3}(L+x)^3 \tag{33}$$

onde $I_A = \frac{bh_0^3}{12}$.

Considerando-se, ainda, o carregamento atuante, define-se o momento fletor atuante no elemento, em função da coordenada x, como segue:

$$M(x) = -Px \tag{34}$$

Substituindo-se as equações (33) e (34) na equação diferencial da linha elástica para elementos fletidos (BEER; JOHNSTON; MAZUREK, 2012), aplicando-se o método de integração direta e substituindo-se as condições de contorno do problema, é possível obter-se a solução analítica para o problema em questão, dada pela equação (35).

$$\upsilon(x) = \frac{PL^3}{I_A}\left[\frac{L}{2(x+L)} - \frac{3x}{8L} + \ln(x+L) - \ln(2L) + \frac{1}{8}\right] \tag{35}$$

Sendo o deslocamento máximo da viga dado por:

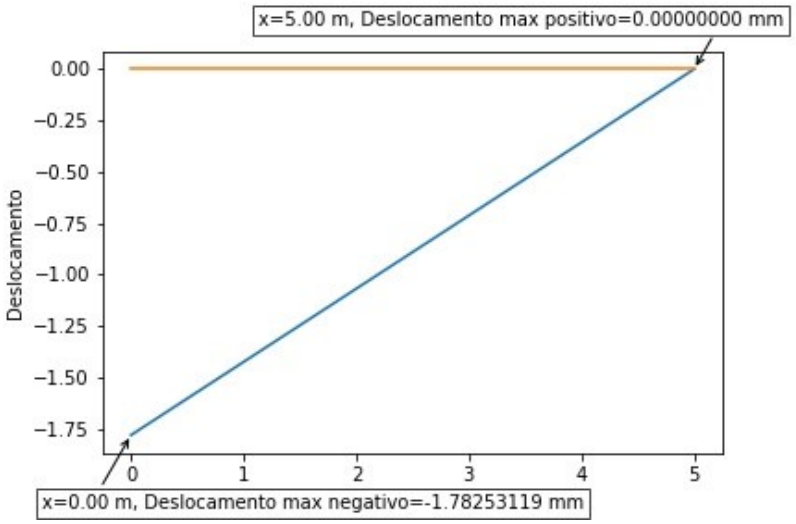
$$\upsilon_{máx} = \upsilon(x=0) = -0,06815\frac{PL^3}{I_A} \tag{36}$$

Por fim, substituindo-se os dados iniciais do problema na equação (36), é possível obter-se o valor de deflexão máxima para a viga em questão:

$$\upsilon_{máx} = -1,81733\times10^{-3}\ \text{m} \tag{37}$$

Inicialmente foi realizada a simulação numérica do problema no software implementado utilizando-se uma malha composta por apenas um elemento finito. O traçado da configuração deformada do elemento estrutural fornecido pelo software pode ser visualizado na figura 19.

Figura 19 – Esboço da configuração deformada do elemento de altura variável.



Fonte: Autoria própria (2021)

Analisando-se os resultados de deslocamento máximo, verifica-se que o código implementado apresentou um resultado igual a $1{,}78253 \text{ x } 10^{-3}\, m$, diferenciando-se da solução analítica na primeira casa decimal e apresentando um erro da ordem de 1,9%.

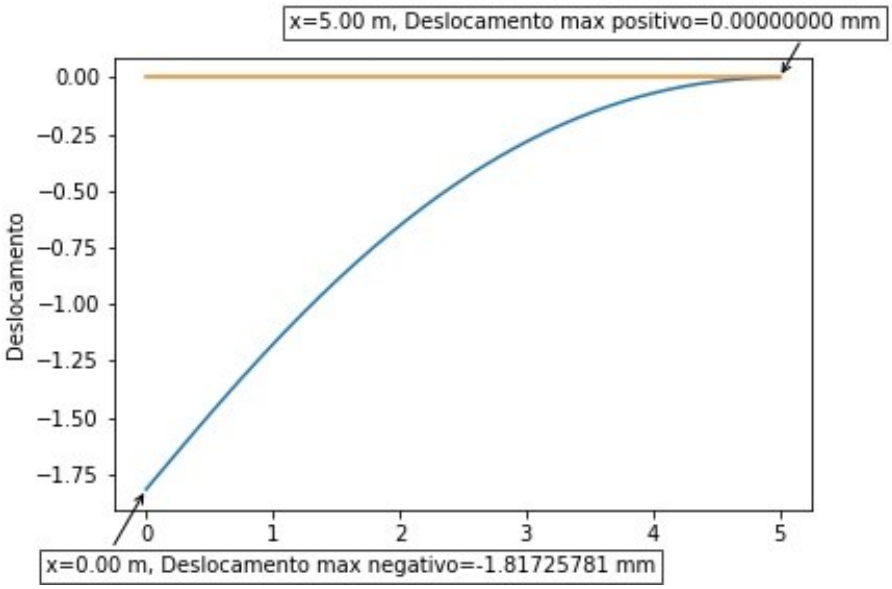
Outras cinco análises foram realizadas variando-se o número de elementos finitos presentes na malha. Os resultados obtidos estão agrupados na Tabela 1. Ainda, considerando-se a malha mais refinada, apresenta-se na figura 20 a configuração deformada do elemento estrutural.

Tabela 1 – Convergência em relação ao número de elementos finitos

Número de elementos finitos	Deslocamento máximo (m)	Erro (%)
1	-0,00178253	1,9
5	-0,00181703	0,02
10	-0,00181724	0,005
50	-0,00181726	0,004
100	-0,00181726	0,004
500	-0,00181726	0,004

Fonte: Autoria própria (2021)

Figura 20 – Configuração deformada do elemento de altura variável com malha refinada em 500 elementos.



Fonte: Autoria própria (2021)

A análise dos resultados apresentados permite concluir que a formulação proposta para análise numérica de vigas retangulares de altura variável via MEF produz resultados confiáveis quando comparados à solução analítica do problema.

Em se tratando da análise das malhas utilizadas nas simulações, percebe-se que a malha composta por apenas um elemento finito produziu solução com apenas 1,9% de erro mesmo utilizando um polinômio interpolador de ordem cúbica não coincidente com a solução analítica do problema. À medida que se utiliza malhas mais refinadas, a solução tende a se estabilizar com erro praticamente nulo.

A exemplo do ocorrido na análise do exemplo descrito neste trabalho, apesar dos resultados nodais satisfatórios, neste trabalho, o traçado mais adequado da curva representativa da configuração deformada do elemento estrutural só é possível utilizando-se malhas mais refinadas.

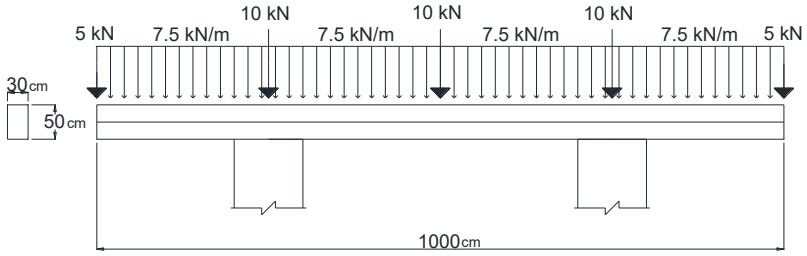
6.3 Exemplo 3

A viga abordada neste exemplo possui 10 metros de comprimento, sendo bi-apoiada e com duas extremidades em balanço. O objetivo é simular uma viga transversina de uma ponte com configuração estrutural conforme apresentado na figura 1 deste trabalho.

Para as simulações realizadas considerou-se a viga composta por um material hipotético com módulo de elasticidade E= $3x10^7$ kN/m². São realizadas simulações numéricas com diversas cargas e variações de altura, comparando os resultados obtidos com os fornecidos pelo software comercial SCIA Engineer versão 20.0 para todos os casos. Nos casos em que se assumiu a viga com altura constante, os resultados foram também comparados com os fornecidos pelo software Ftool.

Inicialmente foi realizada uma análise considerando a transversina com altura constante e sujeita a ação de forças concentradas simulando as vigas longarinas que descarregam nela. Além disso, foi considerado também um carregamento uniformemente distribuído simulando o peso próprio do elemento. O carregamento atuante, bem como a vinculação e as dimensões do problema estão ilustrados na figura 21.

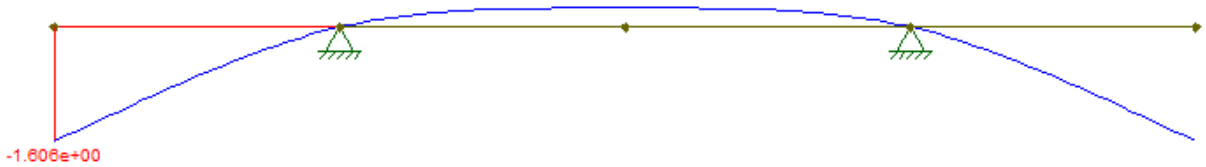
Figura 21 – Viga com carregamentos concentrados e uniformemente distribuído.



Fonte: Autoria própria (2021)

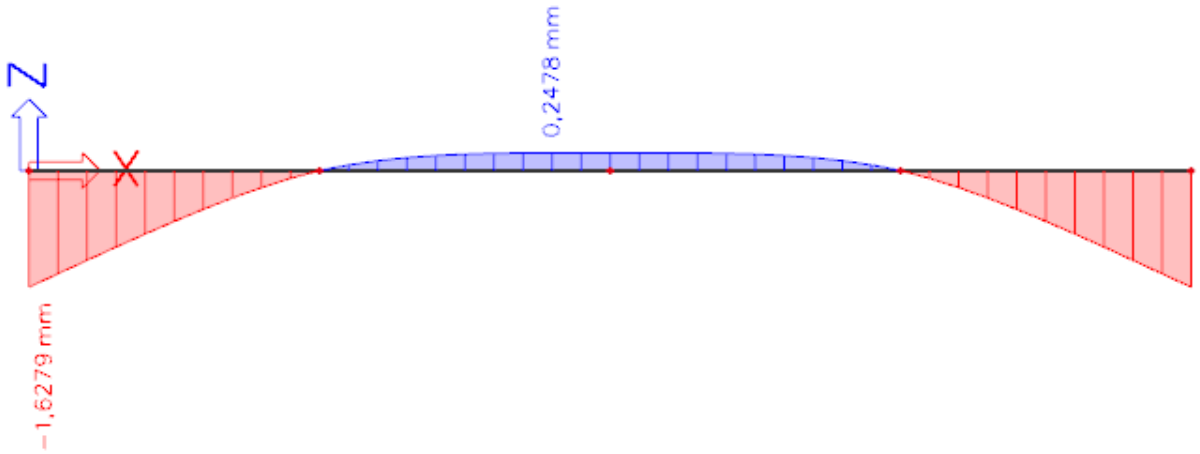
As figuras 22, 23 e 24 apresentam a configuração deformada da viga nas análises realizadas nos softwares Ftool e SCIA Engineer e no software desenvolvido, respectivamente. Observa-se que o software SCIA Engineer possui uma variação de 1,35% em comparação aos resultados de deslocamento máximo obtidos pelo software desenvolvido e pelo software Ftool.

Figura 22 – Configuração deformada obtida pelo Ftool para viga com carregamentos concentrados e uniformemente distribuído.



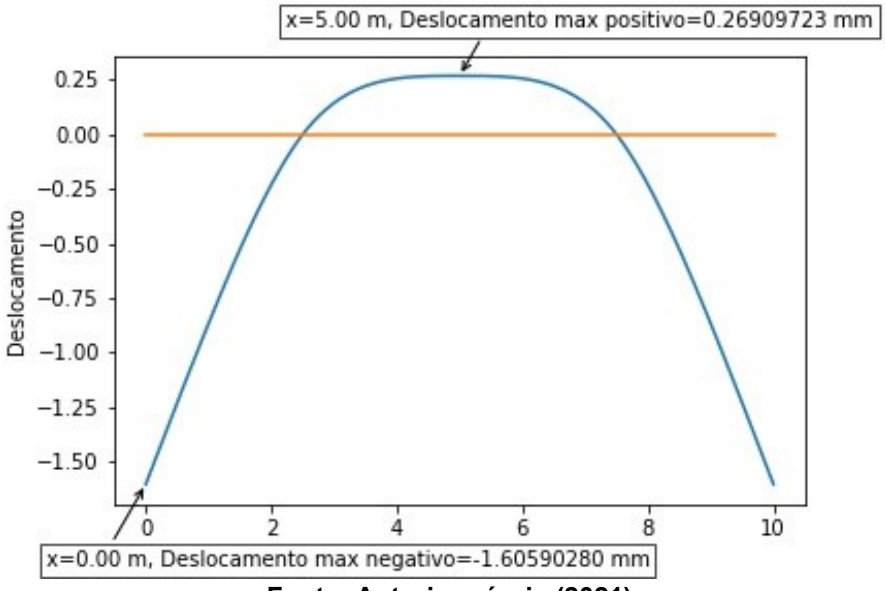
Fonte: Autoria própria (2021)

Figura 23 – Configuração deformada obtida pelo SCIA Engineer para a viga com carregamentos concentrados e uniformemente distribuído.



Fonte: Autoria própria (2021)

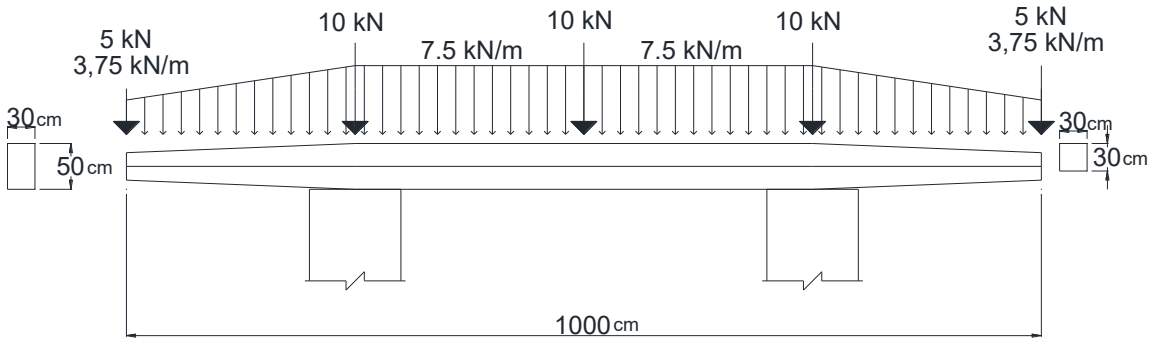
Figura 24 – Configuração deformada obtida pelo software desenvolvido para a viga com carregamentos concentrados e uniformemente distribuído, com malha de 1000 elementos.



Fonte: Autoria própria (2021)

Em uma segunda análise assumiu-se a viga com variação de altura simétrica em relação ao seu eixo longitudinal. Novamente foram considerados os carregamentos concentrados e distribuídos no elemento. Neste último caso, em função da variação de altura nas extremidades em balanço, considerou-se um carregamento trapezoidal simulando o peso próprio do elemento. A figura 25 demonstra a configuração estrutural adotada nesta análise.

Figura 25 – Viga com variação de altura simétrica em relação ao seu eixo longitudinal.

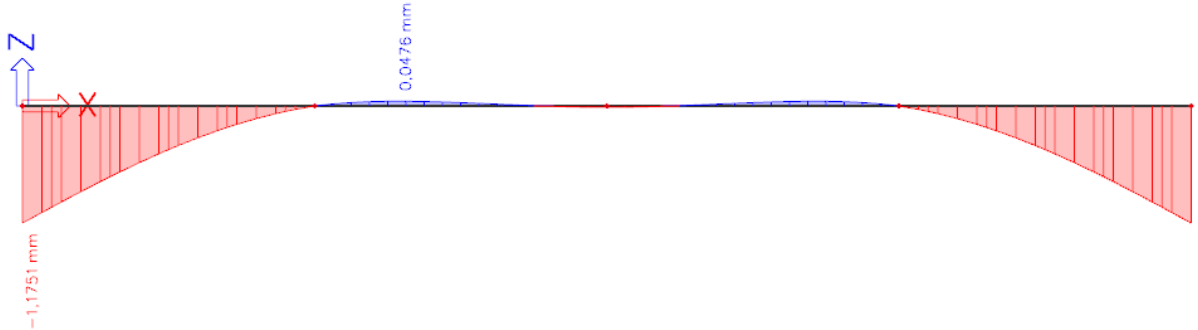


Fonte: Autoria própria (2021)

As figuras 26 e 27 apresentam a configuração deformada da viga nas análises realizadas no software SCIA Engineer e no software desenvolvido, respectivamente. Em função da variação de altura, não foi possível simular o problema no software Ftool.

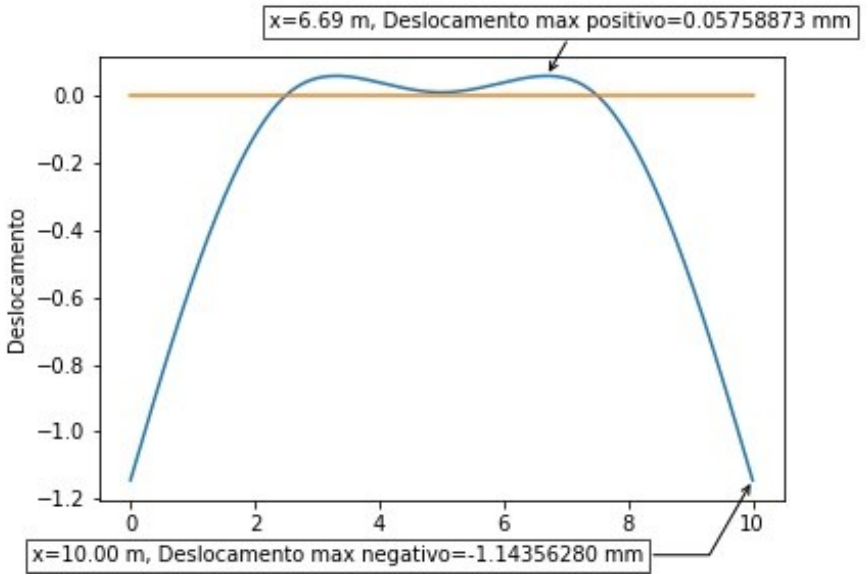
Observa-se que, nesta configuração estrutural, os resultados de deslocamentos máximos obtidos por ambos os softwares apresentam diferenças na ordem de 2,68%.

Figura 26 – Configuração deformada obtida pelo SCIA Engineer para a viga com variação de altura simétrica em relação ao seu eixo longitudinal.



Fonte: Autoria própria (2021)

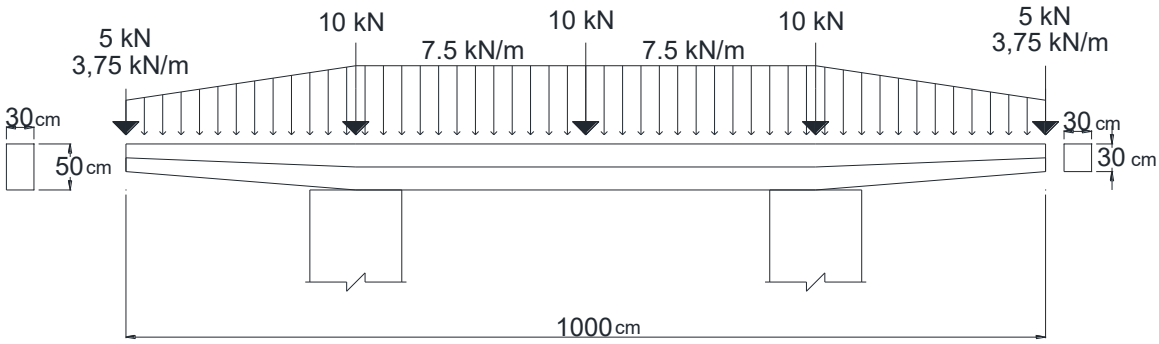
Figura 27 – Configuração deformada obtida pelo software desenvolvido para para a viga com variação de altura simétrica em relação ao seu eixo longitudinal.



Fonte: Autoria própria (2021)

Para finalizar, uma terceira análise foi realizada considerando agora a variação de altura alinhada em relação ao topo da viga. Os carregamentos da segunda simulação foram mantidos bem como as dimensões do problema. A figura 28 ilustra a configuração estrutural adotada nesta análise.

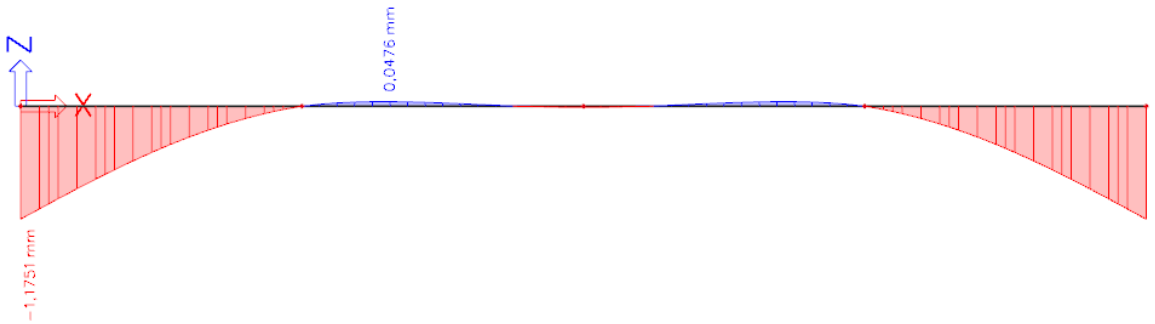
Figura 28 – Viga com variação na altura em relação ao topo.



Fonte: Autoria própria (2021)

A figura 29 ilustra a configuração deformada do elemento fornecida pelo software SCIA Engineer. Uma vez que o software desenvolvido neste trabalho considera apenas a variação de altura simétrica em relação ao eixo longitudinal do elemento, não foi possível realizar esta simulação numérica neste último. O objetivo desta simulação é, então, averiguar a possibilidade do uso da formulação proposta e do software desenvolvido na análise de elementos estruturais com a configuração estrutural aqui apresentada.

Figura 29 – Configuração deformada obtida pelo SCIA Engineer para a viga com variação de altura alinhada ao topo.



Fonte: Autoria própria (2021)

Ao se analisar as figuras 26 e 29 verifica-se que a mudança na variação de altura do elemento estrutural não ocasionou mudanças significativas de deslocamento. Sendo assim, com base neste exemplo, pode-se concluir que o software desenvolvido pode também ser utilizado para a análise de vigas com variação de altura com alinhamento pelo topo.

Por fim, a tabela 2 apresenta uma síntese dos resultados obtidos nas análises propostas neste exemplo enquanto a tabela 3 apresenta uma síntese das diferenças percentuais obtidas a partir da comparação entre os resultados obtidos pelo software desenvolvido e pelos demais softwares mencionados.

Vale ainda estabelecer uma análise comparativa entre as vigas de altura variável e a de altura constante. Os resultados obtidos neste exemplo mostraram que as vigas de altura variável se apresentam menos deformáveis quando comparadas às vigas de altura constante. Apesar de haver uma diminuição da rigidez destes elementos, a diferença do carregamento distribuído em função da redução do peso próprio do elemento acabou por reduzir a magnitude dos deslocamentos estruturais de tais elementos.

Tabela 2 – Dados agrupados do exemplo 3

Caso	Resultado obtido pelo Ftool (mm)	Resultado obtido pelo SCIA Engineer (mm)	Resultado obtido pelo software desenvolvido (mm)
Forças concentradas e carregamento uniformemente distribuído	1,606	1,6279	1,6059
Variação de altura alinhada ao centro	-	1,1751	1,1436
Variação de altura alinhada ao topo	-	1,1751	1,1436

Fonte: Autoria própria (2021)

Tabela 3 – Erro calculado do exemplo 3 agrupado

Caso	Erro em relação ao software Ftool	Erro em relação ao software SCIA Engineer
Forças concentradas e carregamento uniformemente distribuído	0,01%	1,35%
Variação de altura alinhada ao centro	-	2,68%
Variação de altura alinhada ao topo	-	2,68%

Fonte: Autoria própria (2021)

7 CONSIDERAÇÕES FINAIS

O objetivo geral deste trabalho foi elaborar um código computacional a partir do Método dos Elementos Finitos para avaliar o comportamento elástico linear de vigas retangulares com variação linear de altura no seu comprimento, submetidas a variados esforços e condições de contorno.

Utilizando o Princípio dos Trabalhos Virtuais, a Lei de Hooke e outros conceitos da Mecânica dos Sólidos, realizou-se a elaboração da matriz de rigidez e do vetor de cargas equivalentes ao elemento de viga. Para tal, foi necessário a dedução da equação da inércia em relação à altura.

Tendo a matriz de rigidez do elemento, implementou-se um código computacional que, a partir de suas funções de cálculo, é capaz de fornecer a solução do problema e apresentar os diagramas de força cortante e momento fletor, além da configuração deformada da viga em análise em forma de gráficos automatizados, destacando valores de máximo e mínimo e salvando-os em formato de imagem na pasta fonte do software.

Destaca-se que, implantado na linguagem Python, se apresenta de forma que pode ser facilmente compreendido e utilizado, com código fonte disponível neste trabalho e aberto a alterações em futuros estudos relacionados a este tema.

Para a validação do código desenvolvido, apresentou-se neste trabalho três exemplos de casos que sujeitaram o código a diferentes tipos de vigas, testando, assim, cada uma de suas funções. Com os resultados obtidos nos exemplos, realizou-se a comparação com os resultados obtidos em pesquisas de outros autores, soluções analíticas e softwares estruturais existentes.

Pôde-se notar que a implementação do código foi validada uma vez que forneceu resultados precisos para os diagramas de esforços internos e deslocamento estrutural. Destaca-se que a comparação dos resultados obtidos com os fornecidos por softwares e soluções analíticas, pequenas diferenças percentuais foram obtidas, que, em geral, podem ser desprezadas.

Assim, com a experiência obtida ao longo do desenvolvimento deste trabalho, pode-se propor a trabalhos futuros a aplicação desta solução a pórticos em duas dimensões.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR 6118:** Projeto de estruturas de concreto – Procedimento. Rio de Janeiro. 2014.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR 7190:** Projeto de estruturas de madeira**.** Rio de Janeiro. 1997.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR 8681:** Ações e segurança nas estruturas – Procedimento. Rio de Janeiro. 2004.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR 8800:** Projeto de estruturas de madeira. Rio de Janeiro. 2008.

BEER, F. P.; JOHNSTON, E. R.; MAZUREK, D. F. **Mecânica Vetorial para Engenheiros:** Estática. 9 ed. McGraw Hill Brasil, 2019.

BERNARDO, A. R. A. **Elementos estruturais metálicos de inércia variável.** 2012. Dissertação (Mestrado em Engenharia Civil), Universidade de Aveiro, Aveiro, 2012.

BITTENCOURT, M. L., FEIJÓO R. A. **Resistência dos Materiais I.** Notas de aula, Departamento de Engenharia Mecânica, Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2008.

CORRÊA, L. S. **Comparativo de eficiência de vigas de altura constante com vigas de altura variável.** 2013. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Civil), Universidade Federal do Rio Grande do Sul, Porto Alegre, 2013.

HAN, S.; BENAROYA, H.; WEI, T.Dynamics of transversely vibrating beams using four theories. **Journal of Sound and Vibration**, v. 225, n. 5, p. 935-988.1999.

LIMA, P.R. L.; FONTES, C. M. A.; LIMA, J. M. F. Análise não-linear da deflexão de vigas de concreto armado. **Sitientibus**, Feira de Santana, n. 28, p. 91-108, jan./jun. 2003.

LOGAN, D. L. **A First Course in the Finite Element Method.** 4. ed. Platteville: Thomson, 2007.

MOURA, J. Dimensionamento de Viga T. **Guia da Engenharia.** 27 maio 2019. Disponível em: https://www.guiadaengenharia.com/dimensionamento-viga-t/. Acesso em: 19 abr. 2021.

RODRIGUES, L. S. **Estudo e desenvolvimento de código computacional para a avaliação de comportamento estrutural de vigas de Euler-Bernoulli via método dos elementos finitos.** 2016. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Civil) – Universidade Tecnológica Federal do Paraná, Campo Mourão, 2016.

SOUZA, M. F. S. M.; RODRIGUES, R. B.; MASCIA, N. T.**Sistemas estruturais de edificações e exemplos**. Notas de aula, Departamento de Estruturas, Faculdade de de Engenharia Civil, Arquitetura e Urbanismo, Universidade Estadual de Campinas. 2008.

UGURAL, A. C. **Mecânica dos Materiais.** Grupo GEN. 2009.

YUILL, S.; HALPIN, H. **Python**, 2006. Disponível em: http://www.ibiblio.org/hhalpin/homepage/notes/pythonhandout.pdf. Acesso em: 09 dez. 2021.

APÊNDICE A – CÓDIGO FONTE DO PROGRAMA

```
#    Código elaborado para o trabalho de conclusão de curso de Engenharia
Civil
#    Copyright (C) 2021  Thiago Seiji Enokida
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <https://www.gnu.org/licenses/>.

#função para abrir o arquivo de texto para entrada e organização das
variáveis#
def abertura():
    #abre-se o arquivo e busca os dados para alocalos para a maneira a se
trabalhar
    arq= open ("entrada.txt","r")
    #inicia com o vetor inicial, que possui tamanho fixo
    vi=[0,0,0]
    inicial=arq.readline()
    valores=inicial.split()
    vi[0]=int(valores[0])
    vi[1]=int(valores[1])
    vi[2]=float(valores[2])
    #cria-se a matriz para alocar os valores dos nós
    mat_noz = []
    for i in range(0,(vi[0])):
        linha = []
        for j in range(7):
            linha.append(0)
        mat_noz.append(linha)
        #cria-se a matriz para alocar os valores dos elementos
    mat_element = []
    for i in range(0,(vi[1])):
        linha = []
        for j in range(10):
            linha.append(0)
        mat_element.append(linha)
    #insere a numeração na primeira coluna de cada matriz
    for i in range(0,vi[0]):
        mat_noz[i][0]=i+1
    for i in range(0,vi[1]):
        mat_element[i][0]=i+1
    cont=1
    #insere os valores da linha do arquivo em um vetor, que representa uma
das linhas da matriz de nós
    for linha in arq:
        valores = linha.split()
        if cont<=vi[0]:
            v1=float(valores[1])
            v2=int(valores[2])
            v3=int(valores[3])
            v4=int(valores[4])
            v5=int(valores[5])
            v6=float(valores[6])
```

```
            mat_noz[cont-1][1]=v1
            mat_noz[cont-1][2]=v2
            mat_noz[cont-1][3]=v3
            mat_noz[cont-1][4]=v4
            mat_noz[cont-1][5]=v5
            mat_noz[cont-1][6]=v6
            cont+=1
        #insere os valores dos elementos em um vetor que representa a linha
da matriz de elementos
        elif ((cont-vi[0])>=1 and (cont-vi[0])<=vi[1]):
            v1=int(valores[1])
            v2=int(valores[2])
            v3=mat_noz[v1-1][6]
            v4=mat_noz[v1][6]
            v5=float(valores[3])
            v6=float(valores[4])
            v7=int(valores[5])
            v8=float(valores[6])
            v9=mat_noz[v1][1]-mat_noz[v1-1][1]
            mat_element[cont-vi[0]-1][1]=v1
            mat_element[cont-vi[0]-1][2]=v2
            mat_element[cont-vi[0]-1][3]=v3
            mat_element[cont-vi[0]-1][4]=v4
            mat_element[cont-vi[0]-1][5]=v5
            mat_element[cont-vi[0]-1][6]=v6
            mat_element[cont-vi[0]-1][7]=v7
            mat_element[cont-vi[0]-1][8]=v8
            mat_element[cont-vi[0]-1][9]=v9
            cont+=1
    arq.close()
    #retorna a matriz de nós, de elementos, e o coeficiente de detalhamento
inserido
    return mat_noz,mat_element,vi[2]

#divide a matriz em N elementos, de acordo com o que é inserido no
coeficiente de detalhamento
def detalhador(mat_noz,mat_element,det):
    newmat_noz=[]
    newmat_element=[]
    matposição=[]
    C=mat_noz[len(mat_noz)-1][1]
    #insere a primeira linha da nova matriz de elementos, e após isso, a
apaga
    newmat_noz.append(mat_noz[0])
    matposição.append([mat_noz[0][0],mat_noz[0][1]])
    h1=mat_noz[0][6]
    h2=mat_noz[1][6]
    L=mat_noz[1][1]-mat_noz[0][1]
    memoria=mat_noz[0][1]
    mat_noz.pop(0)
    contador=0
    som=1
    #percorre por um contador detalhando e calculando novos nós, levando em
conta de não sobrepor os nós da matriz original, e não pulando elementos
    while contador<(C/det):
        ver1=False
        ver2=False
        vet1=[0,0,0,0,0,0,0]
        if round(contador*det,2)>=mat_noz[0][1]:
            vet1=mat_noz[0]
            h1=mat_noz[0][6]
```

```
                h2=mat_noz[1][6]
                L=mat_noz[1][1]-mat_noz[0][1]
                memoria=mat_noz[0][1]
                mat_noz.pop(0)
                vet1[0]=contador+som
                newmat_noz.append(vet1)
                matposição.append([contador+som,vet1[1]])
                vet1=[0,0,0,0,0,0,0]
                ver1=True
            if round(contador*det,2)!=memoria:
                ver2=True
                xrel=(contador*det)-memoria
                if ver1==True and ver2==True:
                    som+=1
                vet1[0]=(contador+som)
                vet1[1]=(round(contador*det,2))
                vet1[2]=(0)
                vet1[3]=(0)
                vet1[4]=(0)
                vet1[5]=(0)
                vet1[6]=(round(((((h2-h1)/L))*xrel)+h1,15))
                newmat_noz.append(vet1)
            contador+=1
        mat_noz[0][0]=contador+som
        newmat_noz.append(mat_noz[0])
        matposição.append([mat_noz[0][0],mat_noz[0][1]])
        posicionador=0
        #recria a matriz de elementos a partir dos novos nós inseridos no laço
    de repetição anterior
        for i in range(len(newmat_noz)-1):
            if i+1==matposição[posicionador][0]:
                q1=mat_element[posicionador][5]
                q2=mat_element[posicionador][6]
                E=mat_element[posicionador][7]
                b=mat_element[posicionador][8]
                L=mat_element[posicionador][9]
                posicionador+=1
                somador=0
            vet1=[]
            vet1.append(i+1)
            vet1.append(i+1)
            vet1.append(i+2)
            vet1.append(newmat_noz[i][6])
            vet1.append(newmat_noz[i+1][6])
            vet1.append((((q2-q1)/L)*somador)+q1)
            somador+=round(newmat_noz[i+1][1]-newmat_noz[i][1],14)
            vet1.append((((q2-q1)/L)*somador)+q1)
            vet1.append(E)
            vet1.append(b)
            vet1.append(round(newmat_noz[i+1][1]-newmat_noz[i][1],14))
            newmat_element.append(vet1)
        #retorna as novas matrizes que serão utilizadas, caso haja detalhamento
        return newmat_noz,newmat_element


    #recebe as variaveis para criar a matriz de equações para o elemento finito
    def processarmatlocal(v):
        #v=[E,b,L,h1,h2]
        #criação da matriz 4x4
        mat=[]
        for i in range(4):
```

```
        linha = []
        for j in range(4):
            linha.append(0)
        mat.append(linha)
    #inserção dos valores da matriz (previamente descrito no referencial
teórico)

mat[0][0]=((((21*v[3]**3)+(9*(v[3]**2)*v[4])+(9*v[3]*(v[4]**2))+(21*v[4]**3
))*(v[0]*v[1]))/((v[2]**3)*60))

mat[0][1]=((((15*v[3]**3)+(6*(v[3]**2)*v[4])+(3*v[3]*(v[4]**2))+(6*v[4]**3)
)*(v[0]*v[1]))/((v[2]**2)*60))
    mat[0][2]=((((-21*v[3]**3)+(-9*(v[3]**2)*v[4])+(-9*v[3]*(v[4]**2))+(-
21*v[4]**3))*(v[0]*v[1]))/((v[2]**3)*60))

mat[0][3]=((((6*v[3]**3)+(3*(v[3]**2)*v[4])+(6*v[3]*(v[4]**2))+(15*v[4]**3)
)*(v[0]*v[1]))/((v[2]**2)*60))
    mat[1][0]=mat[0][1]

mat[1][1]=((((11*v[3]**3)+(5*(v[3]**2)*v[4])+(2*v[3]*(v[4]**2))+(2*v[4]**3)
)*(v[0]*v[1]))/((v[2])*60))
    mat[1][2]=((((-15*v[3]**3)+(-6*(v[3]**2)*v[4])+(-3*v[3]*(v[4]**2))+(-
6*v[4]**3))*(v[0]*v[1]))/((v[2]**2)*60))

mat[1][3]=((((4*v[3]**3)+(1*(v[3]**2)*v[4])+(1*v[3]*(v[4]**2))+(4*v[4]**3))
*(v[0]*v[1]))/((v[2])*60))
    mat[2][0]=mat[0][2]
    mat[2][1]=mat[1][2]

mat[2][2]=((((21*v[3]**3)+(9*(v[3]**2)*v[4])+(9*v[3]*(v[4]**2))+(21*v[4]**3
))*(v[0]*v[1]))/((v[2]**3)*60))
    mat[2][3]=((((-6*v[3]**3)+(-3*(v[3]**2)*v[4])+(-6*v[3]*(v[4]**2))+(-
15*v[4]**3))*(v[0]*v[1]))/((v[2]**2)*60))
    mat[3][0]=mat[0][3]
    mat[3][1]=mat[1][3]
    mat[3][2]=mat[2][3]

mat[3][3]=((((2*v[3]**3)+(2*(v[3]**2)*v[4])+(5*v[3]*(v[4]**2))+(11*v[4]**3)
)*(v[0]*v[1]))/((v[2])*60))
    for i in range (len(mat)):
        print (mat[i])
    #for i in range (len(newmat_element)):
        #print (newmat_element[i])
    return mat

def matglobal(mat_noz,mat_element):
    #criar vetor de entrada para função matglobal()
    ent=[0,0,0,0,0]
    #cria matriz quadrada com i e j igual a 2 vezes o numero de nós
    matglobal=[]
    for i in range((mat_noz[len(mat_noz)-1][0])*2):
        linha = []
        for j in range((mat_noz[len(mat_noz)-1][0])*2):
            linha.append(0)
        matglobal.append(linha)
    #cria a matriz local para processamento
    matlocal=[]
    for i in range(4):
        linha = []
        for j in range(4):
            linha.append(0)
```

```
            matlocal.append(linha)
        posicionador=0
        #processamento da matriz global
        for i in range(len(mat_element)):
            #aloca os dados necessarios no vetor de entrada

ent=[mat_element[i][7],mat_element[i][8],mat_element[i][9],mat_element[i][3
],mat_element[i][4]]
            #processa os dados do vetor de entrada na função matlocal
            matlocal=processarmatlocal(ent)
            #adiciona o valor da matlocal a sua reespectiva coordenada na
matglobal
            #posicionador é um ponto de partida que avança de 2 em 2 a cada fim
de laço para fixar um novo ponto de partida da matglobal
            for j in range(4):
                for k in range(4):
                    if posicionador<(len(matglobal)):

matglobal[posicionador+j][posicionador+k]+=matlocal[j][k]
            posicionador+=2
        return matglobal

#calcula o vetor de forças, para levar em conta as forças distribuídas
def vetorforças(mat_noz,mat_element):
        vetforças=[]
        vetesf=[0,0,0,0]
        for i in range(mat_noz[len(mat_noz)-1][0]):
            vetforças.append("F")
            vetforças.append("M")
        for i in range(len(mat_noz)):
            vetforças[i*2]=mat_noz[i][4]
            vetforças[(i*2)+1]=mat_noz[i][5]
        for i in range(len(mat_element)):

vetesf[0]=(((7*mat_element[i][9])/20)*(mat_element[i][5]))+(((3*mat_element
[i][9])/20)*(mat_element[i][6]))

vetesf[1]=(((mat_element[i][9]**2)/20)*(mat_element[i][5]))+((((mat_element
[i][9])**2)/30)*(mat_element[i][6]))

vetesf[2]=(((3*mat_element[i][9])/20)*(mat_element[i][5]))+(((7*mat_element
[i][9])/20)*(mat_element[i][6]))

vetesf[3]=((((mat_element[i][9]**2)/30)*(mat_element[i][5]))+((((mat_elemen
t[i][9])**2)/20)*(mat_element[i][6])))*-1
            vetforças[(mat_element[i][1]-1)*2]+=vetesf[0]
            vetforças[((mat_element[i][1]-1)*2)+1]+=vetesf[1]
            vetforças[(mat_element[i][2]-1)*2]+=vetesf[2]
            vetforças[((mat_element[i][2]-1)*2)+1]+=vetesf[3]
        return vetforças

#busca as condições de contorno e atribui zero a linha e as colunas onde
ocorrem as condições
def contorno(matglobal,mat_noz,vet_trat):
        for i in range(len(mat_noz)):
            if mat_noz[i][2]==1:
                vet_trat[i*2]=0.00
                for coluna in range(len(matglobal)):
                    for linha in range(len(matglobal[0])):
                        if coluna==2*i:
                            matglobal[coluna][linha]=0
```

```
                    if linha==2*i:
                        matglobal[coluna][linha]=0
                    if (2*i)==linha and (2*i)==coluna:
                        matglobal[coluna][linha]=1
        if mat_noz[i][3]==1:
            vet_trat[i*2+1]=0.00
            for coluna in range(len(matglobal)):
                for linha in range(len(matglobal[0])):
                    if coluna==(2*i)+1:
                        matglobal[coluna][linha]=0
                    if linha==(2*i)+1:
                        matglobal[coluna][linha]=0
                    if (2*i)+1==linha and (2*i)+1==coluna:
                        matglobal[coluna][linha]=1
    return matglobal,vet_trat

#utiliza-se a função numpy para resolver o sistema de equações com as
devidas condições de contorno já aplicadas
def solução(vettrat,matglobtrat):
    A = np.array(matglobtrat)
    b = np.array(vettrat)
    x = np.linalg.solve(A, b)
    return x

#utiliza-se do vetor de deslocamento e a matriz global para o calculo dos
valores dos apoios
def retorno(matglobal,vetsol):
    vetapoio=[]
    for i in range(len(vetsol)):
        vetapoio.append(0)
    for i in range(len(matglobal)):
        for j in range(len(matglobal[0])):
            vetapoio[i]+=matglobal[i][j]*vetsol[j]
    return(vetapoio)

#subtrai a parcela da força distribuída do vetor que representa as forças
atuantes, assim tendo o vetor de apoio
def distribuida(vetini,vetfin):
    for i in range(len(vetini)):
        vetfin[i]-=vetini[i]
    return vetfin



#calcula os esforços internos e os plota com a função numpy.plot
def esforçosinternos(vetdes,mat_element,mat_noz):
    vetlocal=[[0,0,0,0],[0,0,0,0]]
    vetres=[]
    vetlinha=[]
    #cria uma matriz 4x4 da matriz de rigidez e aplica os valores das
soluções, assim extraindo os valores da força cortante e momento fletor em
cada nó do elemento
    for element in range(len(mat_element)):
        vetres.append([])

ent=[mat_element[element][7],mat_element[element][8],mat_element[element][9
],mat_element[element][3],mat_element[element][4]]
        matlocal=processarmatlocal(ent)
        vetlocal[0][0]=vetdes[((mat_element[element][1]-1)*2)]
        vetlocal[0][1]=vetdes[((mat_element[element][1]-1)*2)+1]
        vetlocal[0][2]=vetdes[((mat_element[element][2]-1)*2)]
```

```
        vetlocal[0][3]=vetdes[((mat_element[element][2]-1)*2)+1]

vetlocal[1][0]=(((7*mat_element[element][9])/20)*(mat_element[element][5]))
+(((3*mat_element[element][9])/20)*(mat_element[element][6]))

vetlocal[1][1]=(((mat_element[element][9]**2)/20)*(mat_element[element][5])
)+((((mat_element[element][9])**2)/30)*(mat_element[element][6]))

vetlocal[1][2]=(((3*mat_element[element][9])/20)*(mat_element[element][5]))
+(((7*mat_element[element][9])/20)*(mat_element[element][6]))

vetlocal[1][3]=((((mat_element[element][9]**2)/30)*(mat_element[element][5]
))+((((mat_element[element][9])**2)/20)*(mat_element[element][6])))*-1
        vetres[element].append(element)
        for i in range(4):
            a=0
            a+=matlocal[i][0]*vetlocal[0][0]
            a+=matlocal[i][1]*vetlocal[0][1]
            a+=matlocal[i][2]*vetlocal[0][2]
            a+=matlocal[i][3]*vetlocal[0][3]
            a-=vetlocal[1][i]
            vetres[element].append(a)
        vetres[element].append(mat_noz[element][1])
        vetres[element].append(mat_noz[element+1][1]-0.0000001)
    for i in range(len(vetres)):
        vetres[i][4]=vetres[i][4]*-1
        vetres[i][3]=vetres[i][3]*-1
    vetplotx=[]
    vetplotyM=[]
    vetplotyC=[]
    for i in range(len(vetres)):
        vetlinha.append(0)
        vetlinha.append(0)
        vetplotx.append(vetres[i][5])
        vetplotx.append(vetres[i][6])
        vetplotyM.append(vetres[i][2])
        vetplotyM.append(vetres[i][4])
        vetplotyC.append(vetres[i][1])
        vetplotyC.append(vetres[i][3])
    vet_coord=[]
    vetlinha2=[]
    for i in range(len(mat_noz)):
        vet_coord.append(mat_noz[i][1])
        vetlinha2.append(0)
    vet_des=[]
    for i in range(len(vetdes)):
        if i%2==0:
            vet_des.append(vetdes[i])
    vetplotx.insert(1,0.0000001)
    vetplotx.insert(len(vetplotx),mat_noz[len(mat_noz)-1][1])
    vetplotyM.insert(0,0)
    vetplotyM.insert(len(vetplotyM),0)
    vetplotyC.insert(0,0)
    vetplotyC.insert(len(vetplotyC),0)
    vetlinha.insert(0,0)
    vetlinha.insert(0,0)
    return
vet_coord,vet_des,vetlinha2,vetplotx,vetplotyM,vetlinha,vetplotyC
```

```
def
plotagem(vet_coord,vet_des,vetlinha2,vetplotx,vetplotyM,vetlinha,vetplotyC)
:
    for i in range(len(vet_des)):
        vet_des[i]=vet_des[i]*1000
    plt.plot(vet_coord,vet_des)
    plt.plot(vet_coord,vetlinha2)
    annot_maxD(vet_coord,vet_des)
    annot_minD(vet_coord,vet_des)
    plt.ylabel('Deslocamento')
    plt.xlabel('Comprimento')
    plt.savefig('deslocamento.jpeg', format='jpeg')
    plt.show()
    plt.plot(vetplotx,vetplotyM)
    plt.plot(vetplotx,vetlinha)
    annot_maxM(vetplotx,vetplotyM)
    annot_minM(vetplotx,vetplotyM)
    plt.ylabel('Momento')
    plt.xlabel('Comprimento')
    plt.savefig('momento fletor.jpeg', format='jpeg')
    plt.show()
    annot_maxC(vetplotx,vetplotyC)
    annot_minC(vetplotx,vetplotyC)
    plt.plot(vetplotx,vetplotyC)
    plt.plot(vetplotx,vetlinha)
    plt.ylabel('Cortante')
    plt.xlabel('Comprimento')
    plt.savefig('força cortante.jpeg', format='jpeg')
    plt.show()

def annot_maxM(x,y, ax=None):
    xmax = x[np.argmax(y)]
    ymax = max(y)
    text= "x={:.2f} m, Momento max={:.3f} kN.m".format(xmax, ymax)
    if not ax:
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="-
>",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="right", va="top")
    ax.annotate(text, xy=(xmax, ymax), xytext=(1.1,1.1), **kw)

def annot_minM(x,y, ax=None):
    xmin = x[np.argmin(y)]
    ymin = min(y)
    text= "x={:.2f} m, Momento min={:.3f} kN.m".format(xmin, ymin)
    if not ax:
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="-
>",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="left", va="top")
    ax.annotate(text, xy=(xmin, ymin), xytext=(-0.1,-0.1), **kw)

def annot_maxC(x,y, ax=None):
    xmax = x[np.argmax(y)]
    ymax = max(y)
    text= "x={:.2f} m, Cortante max={:.3f} kN".format(xmax, ymax)
    if not ax:
```

```
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="-
>",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="right", va="top")
    ax.annotate(text, xy=(xmax, ymax), xytext=(1.1,1.1), **kw)

def annot_minC(x,y, ax=None):
    xmin = x[np.argmin(y)]
    ymin = min(y)
    text= "x={:.2f} m, Cortante min={:.3f} kN".format(xmin, ymin)
    if not ax:
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="-
>",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="left", va="top")
    ax.annotate(text, xy=(xmin, ymin), xytext=(-0.1,-0.1), **kw)

def annot_maxD(x,y, ax=None):
    xmax = x[np.argmax(y)]
    ymax = max(y)
    text= "x={:.2f} m, Deslocamento max positivo={:.8f} mm".format(xmax,
ymax)
    if not ax:
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="-
>",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="right", va="top")
    ax.annotate(text, xy=(xmax, ymax), xytext=(1.1,1.1), **kw)

def annot_minD(x,y, ax=None):
    xmin = x[np.argmin(y)]
    ymin = min(y)
    text= "x={:.2f} m, Deslocamento max negativo={:.8f} mm".format(xmin,
ymin)
    if not ax:
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="-
>",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="left", va="top")
    ax.annotate(text, xy=(xmin, ymin), xytext=(-0.1,-0.1), **kw)


import numpy as np
import matplotlib.pyplot as plt
M1,M2,det=abertura()
if det!=0:
    M1,M2=detalhador(M1,M2,det)
M3=matglobal(M1,M2)
V1=vetorforças(M1,M2)
V2,M4=contorno(M3,M1,V1)
V3=solução(M4,V2)
M3=matglobal(M1,M2)
V4=retorno(M3,V3)
```

```
V1=vetorforças(M1,M2)
V4=distribuida(V1,V4)
V5,V6,V7,V8,V9,V10,V11=esforçosinternos(V3,M2,M1)
plotagem(V5,V6,V7,V8,V9,V10,V11)
```

APÊNDICE B – ENTRADA DO SOFTWARE PARA O EXEMPLO 3

5 4 0.01
1 0 0 0 -500 0 0.3
2 2.5 1 0 -1000 0 0.5
3 5 0 0 -1000 0 0.5
4 7.5 1 0 -1000 0 0.5
5 10 0 0 -500 0 0.3
1 1 2 -375 -750 30000000 0.3
2 2 3 -750 -750 30000000 0.3
3 3 4 -750 -750 30000000 0.3
4 4 5 -750 -375 30000000 0.3