UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO DE CONSTRUÇÃO CIVIL
CURSO DE GRADUAÇÃO DE ENGENHARIA CIVIL

SANDRA MENDES MONZINI ROTTA

O MÉTODO DOS ELEMENTOS FINITOS APLICADO À ANÁLISE LINEAR DE VIGAS DE EULER-BERNOULLI: IMPLEMENTAÇÃO COMPUTACIONAL DO ELEMENTO FINITO UNIDIMENSIONAL COM FUNÇÃO DE APROXIMAÇÃO DE QUINTO GRAU

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO
2014

SANDRA MENDES MONZINI ROTTA

O MÉTODO DOS ELEMENTOS FINITOS APLICADO À ANÁLISE LINEAR DE VIGAS DE EULER-BERNOULLI: IMPLEMENTAÇÃO COMPUTACIONAL DO ELEMENTO FINITO UNIDIMENSIONAL COM FUNÇÃO DE APROXIMAÇÃO DE QUINTO GRAU

Trabalho de Conclusão de Curso, apresentado à Disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior em Engenharia Civil, Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de bacharel em Engenharia Civil.

Orientador: Prof. Dr. Leandro Waidemam

CAMPO MOURÃO

2014

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Campo Mourão
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Construção Civil
Coordenação de Engenharia Civil

TERMO DE APROVAÇÃO

Trabalho de Conclusão de Curso

O MÉTODO DOS ELEMENTOS FINITOS APLICADO À ANÁLISE LINEAR DE VIGAS DE EULER-BERNOULLI: IMPLEMENTAÇÃO COMPUTACIONAL DO ELEMENTO FINITO UNIDIMENSIONAL COM FUNÇÃO DE APROXIMAÇÃO DE QUINTO GRAU

por

Sandra Mendes Monzini Rotta

Este Trabalho de Conclusão de Curso foi apresentado às 15h30min do dia 04 de fevereiro de 2015 como requisito parcial para a obtenção do título de ENGENHEIRO CIVIL, pela Universidade Tecnológica Federal do Paraná. Após deliberação, a Banca Examinadora considerou o trabalho aprovado

Prof. Me. Jeferson Rafael Bueno
(UTFPR)

Prof. Me. Ângelo Giovanni Corelhano
(UTFPR)

Prof. Dr. Leandro Waidemam
(UTFPR)
Orientador

Responsável pelo TCC: **Prof. Me. Valdomiro Lubachevski Kurta**

Coordenador do Curso de Engenharia Civil:
Prof. Dr. Marcelo Guelbert

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

À minha mãe Hildete Alves Monzini (*in memorian*).

AGRADECIMENTOS

Agradeço a Deus por me permitir concluir mais essa etapa de aprendizado e por colocar tantas pessoas boas em minha vida, que me deram força pra chegar até aqui.

Agradeço a toda minha família pelo apoio e confiança, em especial ao meu marido Mauricio e meus filhos Sofia e Eric pela compreensão nas horas de ausência, cansaço e mau humor e por toda a ajuda que me possibilitou concluir esse curso; a meu irmão Wagner e minha prima, de coração, Lara pela substancial ajuda no desenvolvimento desse trabalho.

Agradeço também, a todos os meus amigos e colegas que estudaram comigo e participaram dessa jornada e que, de alguma forma, me ajudaram.

Aos meus professores pelos conhecimentos compartilhados e em especial o meu orientador, Leandro Waidemam, pela paciência (imensa) e por se disponibilizar a me orientar neste e em outros trabalhos.

RESUMO

MONZINI ROTTA, Sandra M. **O Método dos Elementos Finitos aplicado à análise linear de vigas de Euler-Bernoulli: implementação computacional do elemento finito unidimensional com função de aproximação de quinto grau**. 2014. 74 f. Trabalho de Conclusão de Curso – Graduação em Engenharia Civil, Universidade Tecnológica Federal do Paraná. Campo Mourão. 2014.

Este trabalho apresenta um estudo sobre o Método dos Elementos Finitos aplicado à análise elástica linear de vigas de Euler-Bernoulli. O elemento finito apresentado é o elemento de geometria linear, composto por três nós e com polinômio de quinto grau utilizado para aproximar o deslocamento ao longo do elemento a partir de duas variáveis nodais, a saber: deslocamento perpendicular ao seu eixo longitudinal e giro da seção transversal. Com o intuito de validar o modelo proposto, a montagem e resolução do sistema de equações algébricas foram implementados computacionalmente em linguagem Java, dando origem ao *software* JVigas, que tem como principal característica a entrada de dados e saída de resultados (diagramas de estado e estrutura deslocada) de forma gráfica. Ao longo do trabalho são apresentados quatro exemplos numéricos, cujos resultados foram comparados com soluções analíticas, com os fornecidos por outros pesquisadores ou com os obtidos a partir do *software* Ftool. A partir dessas análises, pode-se concluir que o elemento finito apresentado fornece soluções exatas em termos de momento fletor e força cortante, obtidos neste trabalho a partir da equação diferencial da linha elástica e da relação diferencial entre momento fletor e força cortante, para carregamentos concentrados, uniformemente ou linearmente distribuídos ao longo do elemento.

Palavras-chave: Método dos Elementos Finitos. Vigas de Euler-Bernoulli.

.

ABSTRACT

MONZINI ROTTA, Sandra M. **The Finite Element Method applied to the linear analysis of Euler-Bernoulli beams: computational implementation of the unidimensional finite element with fifth-degree shape function.** 2014**.** 74 f. Trabalho de Conclusão de Curso – Graduação em Engenharia Civil, Universidade Tecnológica Federal do Paraná. Campo Mourão. 2014.

This work presents a study of the Finite Element Method applied to the linear elastic analysis of Euler-Bernoulli beams. The finite element presented is the element of linear geometry, composed of three nodes and with a fifth degree polynomial used to approximate the perpendicular displacement along the element from two nodal variables, knowing: the perpendicular displacement to its longitudinal axis and rotation of the cross section. In order to validate the presented model, the assembly and solution of the system of algebraic equations were implemented computationally in Java language, originating the *JVigas* software, which has as main characteristic data input and output (state diagrams and displacements of the structure) in visual form. Throughout the work, four numerical examples are presented, whose results are compared with exact solutions, with the ones provided by other researchers and with the ones obtained from the Ftool software. From these analyses, it can be concluded that the presented finite element provides exact solutions in terms of bending moment and shear force, obtained in this work from the differential equation of the elastic line and the differential relation between bending moment and shear force, for concentrated loads, uniformly or linearly distributed along the element.

Keywords: Finite Element Method. Euler-Bernoulli Beams.

LISTA DE FIGURAS

FIGURA 1 – HIPÓTESES DE BERNOULLI 18
FIGURA 2 – VIGA SUJEITA A CARREGAMENTO LINEAR 19
FIGURA 3 – FORÇAS, MOMENTOS, DESLOCAMENTOS E GIROS NODAIS 20
FIGURA 4 – REPRESENTAÇÃO GRÁFICA DAS FUNÇÕES DE APROXIMAÇÃO. 22
FIGURA 5 – JANELA DE ABERTURA DO PROGRAMA 26
FIGURA 6 – JANELA DE UM NOVO PROJETO 27
FIGURA 7 – JANELA “DETALHES DA VIGA” 28
FIGURA 8 – RESUMO DA VIGA 29
FIGURA 9 – JANELA “DIAGRAMA DA ESTRUTURA DA VIGA” 29
FIGURA 10 – JANELA “REAÇÕES DE APOIO” 30
FIGURA 11 – JANELA DIAGRAMA DE FORÇA CORTANTE 30
FIGURA 12 – JANELA “DIAGRAMA DE MOMENTO FLETOR” 31
FIGURA 13 – JANELA “DESLOCADA DA ESTRUTURA” 31
FIGURA 14 – FLUXOGRAMA DOS PROCESSOS 32
FIGURA 15 – ESTRUTURA DA VIGA 38
FIGURA 16 – VIGA BI APOIADA - CONFIGURAÇÃO DESLOCADA DA ESTRUTURA 39
FIGURA 17 – VIGA BI APOIADA - DIAGRAMAS DE MOMENTOS FLETORES (KN.M) 40
FIGURA 18 – VIGA BI APOIADA - DIAGRAMAS DE FORÇAS CORTANTES (KN) 40
FIGURA 19 – VIGA CONTÍNUA SUJEITA A CARREGAMENTOS DIVERSOS. 42
FIGURA 20 – VIGA CONTÍNUA – MALHA UTILIZADA. 42
FIGURA 21 – VIGA CONTÍNUA – DIAGRAMAS DE MOMENTOS FLETORES (KN.M). 43
FIGURA 22 – VIGA BI APOIADA - DIAGRAMAS DE FORÇAS CORTANTES (KN). 43
FIGURA 23 – VIGA CONTÍNUA UMA VEZ HIPERESTÁTICA, SUJEITA A CARREGAMENTOS DIVERSOS 44
FIGURA 24 – DIAGRAMA DE MOMENTO FLETOR (KN.M) 45
FIGURA 25 – VIGA 6 VEZES HIPERESTÁTICA SUJEITA A CARREGAMENTOS DIVERSOS 46
FIGURA 26 – MALHA UTILIZADA NA ANÁLISE 46
FIGURA 27 – DIAGRAMA DE MOMENTO FLETOR (KN.M) 47

LISTA DE QUADROS

QUADRO 1 – ESTRUTURA DESLOCADA, DIAGRAMAS DE CORPO LIVRE, FORÇA CORTANTE E MOMENTO FLETOR PARA A VIGA BI APOIADA. 39

QUADRO 2 – ESTRUTURA DESLOCADA, REAÇÕES DE APOIO, DIAGRAMAS DE FORÇA CORTANTE E MOMENTO FLETOR PARA A VIGA DE CONCRETO..... 45

QUADRO 4 – REAÇÕES DE APOIO, DIAGRAMAS DE ESTADO E CONFIGURAÇÃO DEFORMADA DA ESTRUTURA. 47

SUMÁRIO

1. INTRODUÇÃO 9
2. OBJETIVOS 11
2.1 OBJETIVO GERAL 11
2.2 OBJETIVOS ESPECÍFICOS 11
3. JUSTIFICATIVA 12
4. REVISÃO BIBLIOGRÁFICA 14
4.1 VIGAS 14
4.2 MÉTODO DOS ELEMENTOS FINITOS 14
4.3 MÉTODO DOS ELEMENTOS FINITOS APLICADO A VIGAS DE EULER-BERNOULLI 16
5. ASPECTOS COMPUTACIONAIS 26
5.1 APRESENTAÇÃO DA INTERFACE GRÁFICA 26
5.1.1 Janela inicial 26
5.1.2 Janela de um novo projeto 26
5.1.3 Janela “Detalhes da viga” 27
5.1.4 Janela “Diagrama da estrutura da viga” 29
5.1.5 Janela “Reações de apoio” 30
5.1.6 Janela “Diagrama de força cortante” 30
5.2 ESQUEMA GERAL DE CÁLCULO 32
5.2.1 Matriz de rigidez local 33
5.2.2 Matriz de rigidez global 33
5.2.3 Vetor de cargas local 33
5.2.4 Vetor de cargas global 34
5.2.5 Condição de contorno 34
5.2.6 Deslocamento 34
5.2.7 Reações de apoio 35
5.2.8 Momento fletor 35
5.2.9 Força cortante 36
5.2.10 Deslocada da estrutura 37
5.2.11 Exibe resultados 37
6 RESULTADOS E DISCUSSÕES 38
6.1 EXEMPLO 1 38
6.2 EXEMPLO 2 41
6.3 EXEMPLO 3 44
6.4 EXEMPLO 4 46
7 CONSIDERAÇÕES FINAIS 49
REFERÊNCIAS 51
APÊNDICE A – Código fonte dos principais processos do JVigas 53

1. INTRODUÇÃO

O engenheiro estrutural precisa desempenhar o seu papel de executar análises e dimensionamentos precisos num período de tempo cada vez menor, garantindo que a estrutura atenda a todos os requisitos de projetos sob as diversas condições de operação, o que requer significativos esforços humanos. Habitualmente são feitas sucessivas análises e modificações das características da estrutura em busca de uma solução ótima, visando economia e segurança aliadas aos pré-requisitos funcionais e regulamentares.

Com o advento do computador esses esforços humanos foram amenizados, uma vez que os profissionais passaram a contar com uma gama de *softwares* que realizam análises estruturais em um tempo reduzido, dispondo ainda de uma interface gráfica. Essa realidade levou o projetista de estruturas a se habilitar ao uso dos respectivos *softwares* e a interpretar corretamente os resultados obtidos, mesmo sem ter acesso ao código fonte do programa ou até mesmo desconhecendo as características do modelo que está utilizando (AZEVEDO, 2003).

O desenvolvimento dos *softwares* citados tem como base a evolução dos métodos numéricos aplicados em problemas de engenharia. Tais métodos permitem obter soluções aproximadas com precisão e custo computacional aceitáveis, sendo variáveis em função das características da estrutura analisada e, principalmente, do modelo mecânico empregado. Dentre os métodos numéricos, o mais utilizado é o Método dos Elementos Finitos (MEF), que é tido como uma técnica numérica adequada para implementação em computadores (UGURAL, 2009).

Atualmente, esse método faz parte da grade curricular de muitos cursos de Engenharia no mundo. O conhecimento do seu funcionamento leva a um uso mais seguro dos programas de análises de estruturas, uma vez que o profissional terá embasamento para perceber eventuais erros na introdução dos dados, saberá escolher o modelo mecânico de acordo com a estrutura a ser analisada, não desprezará condicionantes importantes, etc.

Diante do exposto, o presente trabalho visa desenvolver um algoritmo capaz de avaliar o comportamento elástico linear de vigas de Euler-Bernoulli, embasado no Método dos Elementos Finitos, buscando uma melhor convergência da solução com

mínimo custo computacional, por meio do uso de uma função de aproximação de quinto grau.

2. OBJETIVOS

2.1 OBJETIVO GERAL

Desenvolver um algoritmo baseado no MEF, a partir de elementos finitos unidimensionais com funções de aproximação de quinta ordem, a fim de avaliar o comportamento estrutural de vigas de Euler-Bernoulli.

2.2 OBJETIVOS ESPECÍFICOS

- Abordar teórica e numericamente um modelo para análise elástica-linear de vigas, com base na teoria de flexão de vigas de Euler-Bernoulli e no MEF;
- Desenvolver um algoritmo que contemple as diversas possibilidades de análise estática elástica e linear de vigas;
- Analisar o comportamento elástico linear de vigas submetidas a carregamentos diversos.

3. JUSTIFICATIVA

O estudo e desenvolvimento dos métodos numéricos aplicados na solução de problemas de engenharia, mais especificamente na modelagem do comportamento mecânico dos sólidos deformáveis, vêm de longa data. A difusão de tais métodos tem como principal justificativa a possibilidade de obtenção, através de procedimentos aproximados, de soluções para problemas complexos dentro de uma precisão considerada aceitável.

Dentre os métodos numéricos existentes, o MEF se destaca por permitir uma completa automatização dos cálculos, justificando sua ampla aplicação em *softwares* computacionais de análise estrutural. Além disso, permite descrever facilmente a geometria da estrutura e possibilita a consideração de modelos mecânicos mais completos, sendo aplicável na análise de elementos compostos por materiais compósitos e/ou anisotrópicos, em condições de cargas combinadas, sob carregamentos térmicos e dinâmicos e ainda em problemas estruturais não-lineares.

Nesse contexto, o presente projeto de Trabalho de Conclusão de Curso visa contribuir para o estudo dos métodos numéricos aplicados em análise estrutural por meio de uma proposta de formulação do MEF para a modelagem de vigas de Euler-Bernoulli.

Por serem bastante comuns nas edificações, a descrição do comportamento mecânico das vigas é de especial interesse, principalmente no que diz respeito à obtenção dos esforços internos (força cortante e momento fletor) e de flechas máximas, por se tratarem de informações imprescindíveis no processo de dimensionamento estrutural de tais elementos.

Existem duas hipóteses cinemáticas comuns para representar vigas: a hipótese de Euler-Bernoulli e a de Timoshenko. Basicamente, o que as difere é o fato de que, nesta última, é levado em consideração a deformação por esforço cortante, o que é mais adequado para vigas altas, com relação comprimento/altura menor que 5, sob valores elevados desse esforço (UGURAL, 2009). Como esse tipo de viga é pouco usual, optou-se neste trabalho pela modelagem utilizando teoria de flexão de vigas de Euler-Bernoulli.

No que diz respeito à formulação do MEF proposta, neste trabalho optou-se pela discretização estrutural através de elementos finitos de geometria linear,

compostos por três nós e duas variáveis nodais a saber: deslocamento perpendicular ao eixo longitudinal e giro no plano. Tal configuração permite a utilização de funções de aproximação representadas por polinômios de quinto grau, o que leva à soluções precisas com malhas reduzidas para vigas submetidas à carregamentos concentrados, uniformemente e linearmente distribuídos.

4. REVISÃO BIBLIOGRÁFICA

4.1 VIGAS

A viga é um elemento estrutural retilíneo sujeito a flexão força cortante. Carregamentos externos atuam perpendicularmente ao eixo da peça (CASTRO, 2009), o que desenvolve esforços internos de momentos fletores e forças cortantes. Ocasionalmente podem também estar sujeitos a esforço normal e a torção.

Na teoria de Euler-Bernoulli as distorções devidas à força cortante são desprezadas frente às deformações ocasionadas pelo momento fletor. O campo de deslocamentos é representado pelo deslocamento transversal e pelo ângulo de rotação da seção transversal, em relação ao eixo longitudinal da peça (UGURAL, 2009).

A teoria de Timoshenko para flexão de vigas admite a hipótese de que seções planas perpendiculares ao seu eixo em sua posição indeformada permanecem planas após a deformação mas não necessariamente perpendicular ao eixo. Essa variação do ângulo formado entre a seção transversal e o eixo da peça é relacionada à distorção angular devido ao efeito de cisalhamento (γ) (CASTRO, 2009).

4.2 MÉTODO DOS ELEMENTOS FINITOS

O desenvolvimento inicial do Método dos Elementos Finitos (MEF) é atribuído à indústria aeroespacial, dos Estados Unidos e do Reino Unido, nos anos de 1950, embora não tenham usado o termo elementos finitos. Foi resultado da evolução do computador e da Análise Matricial de Estruturas. Intuitivamente foi feito uma abordagem, utilizando o Princípio dos Trabalhos Virtuais sem o conhecimento de critérios de convergência. (FISH, 2009; SORIANO, 2009).

O MEF é uma aproximação numérica onde equações diferenciais parciais podem ser resolvidas de forma aproximada. Nesse modelo, uma estrutura é dividida, ou discretizada, em partes finitas, chamadas de elementos, interligadas por nós; esse conjunto é denominado malha. Quanto maior for a malha, maior será o esforço

computacional para resolução do problema e tendem a ser a solução exata das equações diferenciais parciais quando o número de nós tenderem ao infinito. (FISH, 2009).

Outro conceito importante é o de graus de liberdade. Originado no movimento das partículas e que diz que um ponto, no espaço tridimensional, apresenta três graus de liberdade sendo esses os três possíveis movimentos de translação que o elemento pode sofrer (SOUZA, 2003). Para corpos rígidos representados no espaço tridimensional, são definidos até sete graus de liberdade, sendo três translações e três rotações e empenamento (BEER E JHONSTON, 1994).

Para qualquer forma de estrutura ou tipo de carregamento, a configuração deformada da estrutura é determinada pelos deslocamentos dos nós. Desta forma, é a partir dos deslocamentos nodais que serão determinados os esforços internos (ALVES FILHO, 2000).

Quando se utiliza um modelo discretizado são considerados apenas os deslocamentos dos nós do modelo, para a representação da configuração deformada da estrutura inteira. As cargas também devem atuar somente nos nós, de forma discreta utilizando-se do conceito de Cargas Nodais Equivalentes, que são cargas que produziriam o mesmo comportamento global na estrutura que as cargas distribuídas nos vãos dos elementos, mesmo atuando somente nos nós do modelo. Desta forma, determinar a relação entre as cargas que atuam nos nós e os seus respectivos deslocamentos em toda a estrutura é o grande intuito da análise estrutural, ou seja, conhecer a rigidez (K) da estrutura (ALVES FILHO, 2000).

A distribuição das cargas/deslocamentos nos pontos restantes (não nodais) do elemento é feita por meio de uma função de interpolação que usa os valores das variáveis de campo obtidos nos nós para fazer uma aproximação ao longo do elemento (CAMPILHO, 2012).

Soriano (2009) ressalta a importância do estudo do MEF, pois, com ele é possível analisar praticamente o comportamento de qualquer sistema físico, tornando-se uma das mais eficientes ferramentas numérica de resolução de equações diferenciais com condições de contorno.

A aplicação do MEF cresceu de forma considerável entre diversos campos em que é possível sua aplicação (REMO, 2003), uma vez que qualquer fenômeno

governado por uma equação diferencial pode ser analisado como MEF. Dentre eles a indústria da construção civil, automobilística, naval, aeronáutica e aeroespacial, telecomunicações, meio ambiente e outros.

Campilho (2012) ressalta que a diversidade de *softwares* baseados no MEF que existem atualmente no mercado é devido a sua grande aplicação em diversas áreas para análise, projeto ou investigação e ao potencial de modelagem numérica. E cita os que mais se destacam atualmente como sendo Abaqus, Ansys, Nastram, SAP2000 e Robot.

4.3 MÉTODO DOS ELEMENTOS FINITOS APLICADO A VIGAS DE EULER-BERNOULLI

O princípio de conservação de energia estabelece que a energia total em um sistema isolado e conservativo permanece constante. Para o caso particular da engenharia de estruturas, o trabalho realizado pelas forças externas atuantes em um elemento estrutural é convertido em energia interna de deformação à medida que este se deforma.

Baseado nesse princípio, John Bernoulli em 1717 desenvolveu o Princípio dos Trabalhos Virtuais (PTV) que permite determinar os deslocamentos dos sistemas elásticos devidos às forças aplicadas (POPOV, 1978).

O PTV estabelece que, a um corpo sujeito à ação de várias forças e em condição de equilíbrio estático, uma vez atribuído um deslocamento virtual (valor imaginário imposto sobre um sistema estrutural em qualquer direção), o trabalho virtual realizado pelas forças reais externas em função do deslocamento virtual é igual ao trabalho virtual realizado pelas forças internas reais em função das deformações virtuais (TIMOSHENKO, 1984).

Destaca-se que a validade do princípio está diretamente ligada à necessidade de satisfação das condições de equilíbrio e compatibilidade, ou seja, as cargas externas relacionadas unicamente aos esforços internos e os deslocamentos unicamente relacionados às deformações internas (HIBBELER, 2004).

A equação (1) descreve matematicamente o princípio estabelecido:

$$U^*_i = U^*_e \tag{1}$$

em que U^*_i representa o trabalho virtual das forças internas e U^*_e o trabalho virtual das forças externas.

Particularizando-se o desenvolvimento para apenas um elemento finito de viga e considerando-se um campo de deslocamentos e deformações virtuais, o trabalho virtual interno pode ser equacionado como segue:

$$U^*_i = \int_V \sigma.\delta\varepsilon^* dV + \int_V \tau.\delta\gamma^* dV \tag{2}$$

onde: σ é a tensão normal; $\delta\varepsilon^*$ deslocamento devido a deformação específica virtual; τ é a tensão de cisalhamento e $\delta\gamma^*$ é o deslocamento devido a deformação por cisalhamento virtual.

Conforme estabelecido anteriormente, a opção por trabalhar com a teoria de Euler-Bernoulli permite que na equação (2) seja desprezada a parcela de energia proveniente das tensões de cisalhamento (UGURAL, 2009). Dessa forma, pode-se escrever:

$$U^*_i = \int_V \sigma.\delta\varepsilon^* dV \tag{3}$$

As deformações específicas de um elemento fletido, definidas a partir da equação diferencial da linha elástica (BEER E JHONSTON, 1996), podem ser descritas como segue:

$$\varepsilon = \left(\frac{d^2\upsilon}{dx^2} \cdot y \right) \tag{4}$$

onde υ representa o campo de deslocamentos na direção vertical y definido a partir do centroide da seção transversal da viga (Figura 1).

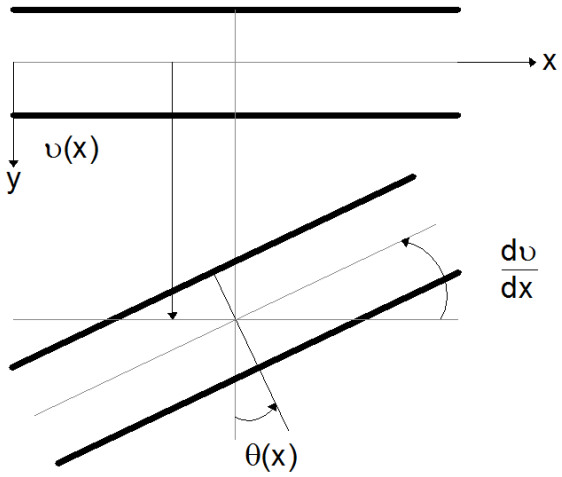


Figura 1: Hipóteses de Bernoulli
Fonte: Adaptado de Castro (2009, p. 14).

Para materiais com comportamento elástico-linear, as tensões normais se relacionam com as deformações específicas através da lei de Hooke (UGURAL, 2009). Considerando-se ainda as deformações descritas na equação (4), é possível escrever:

$$\sigma = E \cdot \varepsilon = E \cdot \left(\frac{d^2 \upsilon}{dx^2} \cdot y \right) \tag{5}$$

sendo E o módulo de elasticidade longitudinal do material.

Substituindo-se as equações (4) e (5) em (3), tem-se:

$$U^*_i = \int_0^V E \cdot \left(\frac{d^2 \upsilon}{dx^2} \cdot y \right) \cdot \left(\frac{d^2 \delta \upsilon^*}{dx^2} \cdot y \right) \cdot dV \tag{6}$$

A integral de volume representada em (6) pode ser convertida em uma integral dupla na área e no comprimento do elemento conforme segue:

$$U^*_I = E \int_0^L \left(\frac{d^2 \upsilon}{dx^2} \right) \cdot \left(\frac{d^2 \upsilon^*}{dx^2} \right) \int_A y^2 \cdot dA \cdot dx \tag{7}$$

sendo a integral na área representativa do momento de inércia em relação ao eixo horizontal neutro da viga, conforme descrito na equação (8).

$$I_z = \int_A y^2 \cdot dA \tag{8}$$

Considerando-se a área de seção transversal constante ao longo do comprimento do elemento, tem-se a expressão final que denota o trabalho virtual interno em função de um campo de deslocamentos e deformações virtuais:

$$U_I^* = EI \int_0^L \left(\frac{d^2\upsilon}{dx^2} \right) \cdot \left(\frac{d^2\upsilon^*}{dx^2} \right) \cdot dx \tag{9}$$

A função $\upsilon(x)$ representativa da linha elástica de elementos fletidos deve ser contínua ao longo de todo o comprimento do elemento. Além disso, ela pode ser representada por uma equação diferencial de, no mínimo, quarta ordem em função do carregamento distribuído atuante no elemento (BEER E JHONSTON, 1996). Tal representação é apresentada na equação (10) que segue:

$$EI \frac{d^4\upsilon}{dv^4} = -q(x) \tag{10}$$

em que $q(x)$ é o carregamento distribuído no comprimento do elemento (Figura 2).

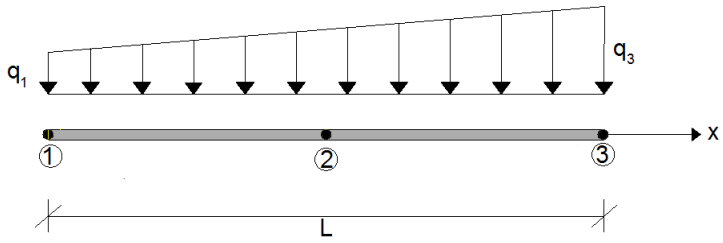


Figura 2: Viga sujeita a carregamento linear
Fonte: Autoria própria.

Neste trabalho, optou-se por trabalhar com carregamentos distribuídos representados por, no máximo, funções lineares, pois, esse tipo de carregamento é adequado para representar a maioria dos casos de carga distribuída. Sendo assim, a partir da equação (10), pode-se definir representativamente:

$$\frac{d^4\upsilon}{dv^4} = a \cdot x + b \tag{11}$$

A solução da equação de governo é admitida como um polinômio do quinto grau na coordenada x, ou seja:

$$\upsilon(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 + \alpha_4 x^4 + \alpha_5 x^5 \tag{12}$$

Para a obtenção do polinômio $\upsilon(x)$ é necessário o conhecimento de seis condições de contorno ao longo do elemento finito. Essas condições de contorno serão definidas a partir dos deslocamentos e giros nodais conforme definido na metodologia do MEF.

Diante do exposto, neste trabalho será utilizado um elemento finito unidimensional (Figura 3), composto por três nós e duas variáveis nodais, a saber: deslocamento vertical υ e giro da seção θ. A cada deslocamento vertical estará associado uma força nodal equivalente F e aos giros, momentos nodais equivalentes M. Tais forças e momentos nodais serão obtidos a partir do carregamento distribuído no elemento no decorrer deste trabalho. O deslocamento axial será desconsiderado.

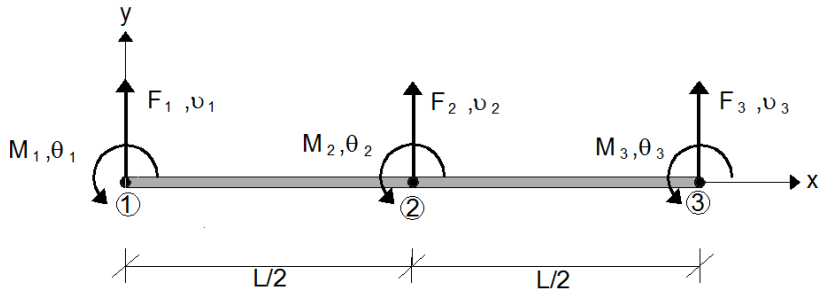


Figura 3: Forças, momentos, deslocamentos e giros nodais
Fonte: Adaptado de Ugural (2009, p. 572).

Sendo assim, lembrando-se que o ângulo de giro $\theta = \frac{d\upsilon}{dx}$ e a partir dos deslocamentos e giros nodais definidos, a equação (12) pode ser definida como segue:

$$\begin{aligned}\upsilon(x) = \upsilon_1 + \theta_1 x + \left(\frac{-23\upsilon_1}{L^2} + \frac{16\upsilon_2}{L^2} + \frac{7\upsilon_3}{L^2} - \frac{6\theta_1}{L} - \frac{8\theta_2}{L} + \frac{\theta_3}{L}\right)x^2 + \\ + \left(\frac{66\upsilon_1}{L^3} - \frac{32\upsilon_2}{L^3} - \frac{34\upsilon_3}{L^3} + \frac{13\theta_1}{L^2} + \frac{32\theta_2}{L^2} + \frac{5\theta_3}{L^2}\right)x^3 + \\ + \left(\frac{-68\upsilon_1}{L^4} + \frac{16\upsilon_2}{L^4} + \frac{52\upsilon_3}{L^4} - \frac{12\theta_1}{L^3} - \frac{40\theta_2}{L^3} - \frac{8\theta_3}{L^3}\right)x^4 + \\ + \left(\frac{24\upsilon_1}{L^5} - \frac{24\upsilon_3}{L^5} + \frac{4\theta_1}{L^4} + \frac{16\theta_2}{L^4} + \frac{4\theta_3}{L^4}\right)x^5\end{aligned} \tag{13}$$

Que reordenada em função das variáveis nodais assume a forma:

$$\begin{aligned}\upsilon(x) = \left(1 - \frac{23x^2}{L^2} + \frac{66x^3}{L^3} - \frac{68x^4}{L^4} + \frac{24x^5}{L^5}\right)\upsilon_1 + \\ + \left(x - \frac{6x^2}{L} + \frac{13x^3}{L^2} - \frac{12x^4}{L^3} + \frac{4x^5}{L^4}\right)\theta_1 + \left(\frac{16x^2}{L^2} - \frac{32x^3}{L^3} + \frac{16x^4}{L^4}\right)\upsilon_2 + \\ + \left(\frac{-8x^2}{L} + \frac{32x^3}{L^2} - \frac{40x^4}{L^3} + \frac{16x^5}{L^4}\right)\theta_2 + \left(\frac{7x^2}{L^2} - \frac{34x^3}{L^3} + \frac{52x^4}{L^4} - \frac{24x^5}{L^5}\right)\upsilon_3 + \\ + \left(-\frac{x^2}{L} + \frac{5x^3}{L^2} - \frac{8x^4}{L^3} + \frac{4x^5}{L^4}\right)\theta_3\end{aligned} \tag{14}$$

E na forma simplificada:

$$\upsilon(x) = \phi_1\upsilon_1 + \phi_2\theta_1 + \phi_3\upsilon_2 + \phi_4\theta_2 + \phi_5\upsilon_3 + \phi_6\theta_3 \tag{15}$$

As funções $\phi_i(x)$ presentes em (15) possui como significado físico o campo de deslocamentos $\upsilon(x)$ (ou rotação $\theta(x)$) que se desenvolve no elemento finito que se impõe o deslocamento nodal (ou giro nodal) independente i e se garante que os restantes sejam nulos (CASTRO, 2009). A representação gráfica dessas funções em relação a coordenada ξ, sendo $\xi = \frac{x}{L}$, está ilustrada na figura 4.

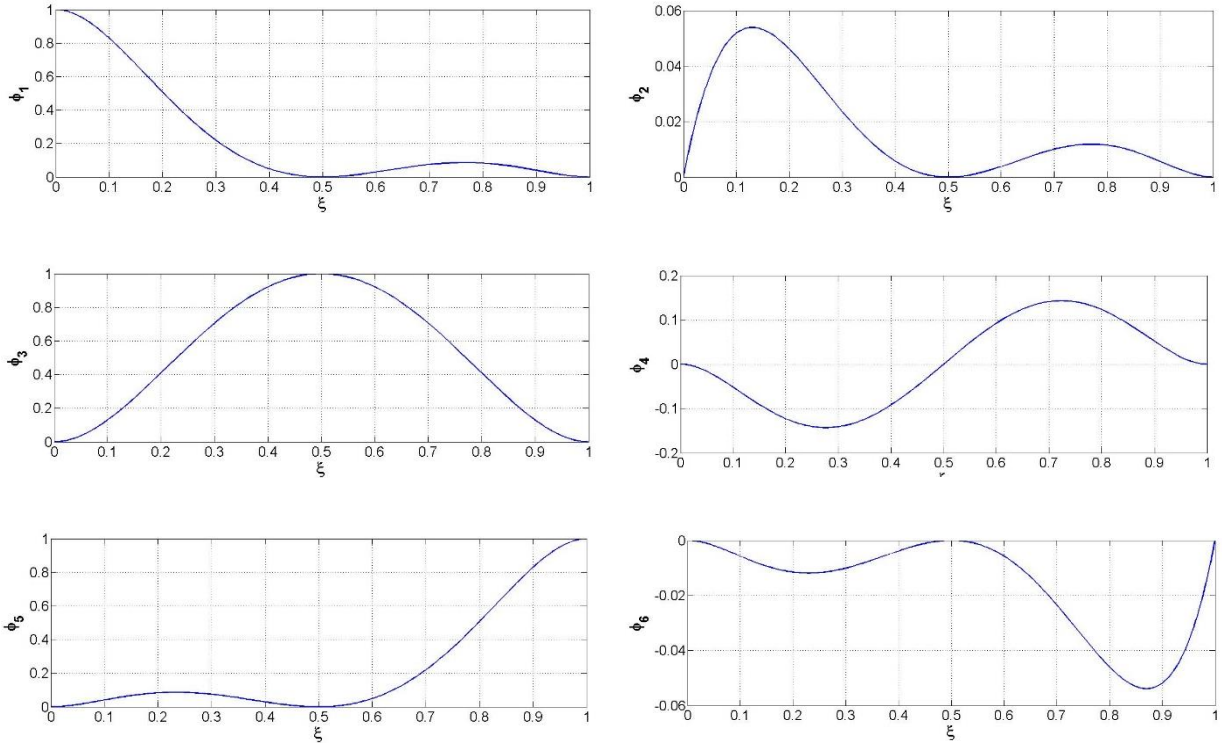


Figura 4: Representação gráfica das funções de aproximação
Fonte: Autoria própria.

Substituindo a função deslocamento (15) em (9) e utilizando-se da mesma função de aproximação para aproximar o campo de deslocamentos virtuais, obtém-se:

$$U_i^* = EI\int_0^L \left(\frac{d^2\phi_1}{dx^2}\right)\left(\frac{d^2\phi_1}{dx^2}\right)\upsilon_1\cdot\upsilon_1^* + \left(\frac{d^2\phi_1}{dx^2}\right)\left(\frac{d^2\phi_2}{dx^2}\right)\upsilon_1\cdot\theta_1^* + \cdots + \left(\frac{d^2\phi_6}{dx^2}\right)\left(\frac{d^2\phi_6}{dx^2}\right)\theta_3\cdot\theta_3^*\cdot dx \quad (16)$$

Atribuindo a

$$\varphi_{ij} = EI\int_0^L \left(\frac{d^2\phi_i}{dx^2}\right)\left(\frac{d^2\phi_j}{dx^2}\right)\cdot dx \quad (17)$$

E reorganizando equação (16) na forma matricial, obtém-se:

$$U_i^* = EI \begin{bmatrix} \varphi_{11} & \varphi_{12} & \varphi_{13} & \varphi_{14} & \varphi_{15} & \varphi_{16} \\ \varphi_{21} & \varphi_{22} & \varphi_{23} & \varphi_{24} & \varphi_{25} & \varphi_{26} \\ \varphi_{31} & \varphi_{32} & \varphi_{33} & \varphi_{34} & \varphi_{35} & \varphi_{36} \\ \varphi_{41} & \varphi_{42} & \varphi_{43} & \varphi_{44} & \varphi_{45} & \varphi_{46} \\ \varphi_{51} & \varphi_{52} & \varphi_{53} & \varphi_{54} & \varphi_{55} & \varphi_{56} \\ \varphi_{61} & \varphi_{62} & \varphi_{63} & \varphi_{64} & \varphi_{65} & \varphi_{66} \end{bmatrix} \begin{bmatrix} \upsilon_1 \\ \theta_1 \\ \upsilon_2 \\ \theta_2 \\ \upsilon_3 \\ \theta_3 \end{bmatrix} \begin{bmatrix} \delta\upsilon_1^* \\ \delta\theta_1^* \\ \delta\upsilon_2^* \\ \delta\theta_2^* \\ \delta\upsilon_3^* \\ \delta\theta_3^* \end{bmatrix}^T \tag{18}$$

Na equação (18), o coeficiente φ_{ij} representa a força no grau de liberdade i devido ao deslocamento unitário imposto ao grau de liberdade j, mantendo fixos os outros graus de liberdade (ALVES FILHO, 2000).

A matriz contida na equação (18) é definida como sendo a matriz de rigidez do elemento k.

$$k = EI \begin{bmatrix} \frac{5092}{35L^3} & \frac{1138}{35L^2} & \frac{-3584}{35L^3} & \frac{1920}{35L^2} & \frac{-1508}{35L^3} & \frac{242}{35L^2} \\ \frac{1138}{35L^2} & \frac{332}{35L} & \frac{-896}{35L^2} & \frac{320}{35L} & \frac{-242}{35L^2} & \frac{38}{35L} \\ \frac{-3584}{35L^3} & \frac{-896}{35L^2} & \frac{7168}{35L^3} & 0 & \frac{-3584}{35L^3} & \frac{896}{35L^2} \\ \frac{1920}{35L^2} & \frac{320}{35L} & 0 & \frac{1280}{35L} & \frac{-1920}{35L^2} & \frac{320}{35L} \\ \frac{-1508}{35L^3} & \frac{-242}{35L^2} & \frac{-3584}{35L^3} & \frac{-1920}{35L^2} & \frac{5092}{35L^3} & \frac{-1138}{35L^2} \\ \frac{242}{35L^2} & \frac{38}{35L} & \frac{896}{35L^2} & \frac{320}{35L} & \frac{-1138}{35L^2} & \frac{332}{35L} \end{bmatrix} \tag{19}$$

Retornando na equação (1), de forma a garantir o equilíbrio de energia, o trabalho virtual externo deve ser considerado. Assim, considerando-se o carregamento distribuído trapezoidal atuando sobre o elemento (figura 2), é possível equacionar o trabalho externo como segue:

$$U_e^* = \int_0^L q(x) \cdot \delta u^*(x) \cdot dx \tag{20}$$

A força que age distribuída no elemento deve ser substituídas por forças nodais equivalentes de forma que o conjunto de cargas obtidos seja estaticamente equivalente ao carregamento distribuído. Assim, “assegurar essa equivalência, as forças nodais equivalentes são determinadas de forma a garantir, qualquer que seja a deformada considerada, que realizam o mesmo trabalho que as forças distribuídas que substituem”(CASTRO, 2009).

Sendo assim, retornando na equação (20) e substituindo o campo de deslocamentos virtuais pela aproximação obtida em (15), obtém-se:

$$U_e^* = \int_0^L q(x)\cdot\left(\phi_1\delta\upsilon_1^* + \phi_2\delta\theta_1^* + \phi_3\delta\upsilon_2^* + \phi_4\delta\theta_2^* + \phi_5\delta\upsilon_3^* + \phi_6\delta\theta_3^*\right)\cdot dx \tag{21}$$

e sendo a função $q(x)$ definida por

$$q(x) = \left(\frac{q_3 - q_1}{L}\right)x + q_1 \tag{22}$$

tem-se como resultado final:

$$U_e^* = \frac{L^2}{420}\begin{bmatrix} \frac{1}{L}(79q_1 + 19q_3) \\ (5q_1 + 2q_3) \\ \frac{112}{L}(q_1 + q_3) \\ (8q_3 - 8q_1) \\ \frac{1}{L}(19q_1 + 79q_3) \\ -(2q_1 + 5q_3) \end{bmatrix}\begin{bmatrix} \delta\upsilon_1^* \\ \delta\theta_1^* \\ \delta\upsilon_2^* \\ \delta\theta_2^* \\ \delta\upsilon_3^* \\ \delta\theta_3^* \end{bmatrix}^T \tag{23}$$

O vetor definido a partir das características do carregamento em (23) é denominado de vetor de forças nodais equivalentes. Vale lembrar que, na eventualidade de forças e momentos concentrados nos nós, estes devem ser somados às linhas correspondentes do referido vetor.

Substituindo as equações (18) e (23) em (1) e lembrando-se que os deslocamentos virtuais são arbitrários e não nulos, tem-se o sistema de equações algébricas para um elemento finito:

$$k = EI\begin{bmatrix} \frac{5092}{35L^3} & \frac{1138}{35L^2} & \frac{-3584}{35L^3} & \frac{1920}{35L^2} & \frac{-1508}{35L^3} & \frac{242}{35L^2} \\ \frac{1138}{35L^2} & \frac{332}{35L} & \frac{-896}{35L^2} & \frac{320}{35L} & \frac{-242}{35L^2} & \frac{38}{35L} \\ \frac{-3584}{35L^3} & \frac{-896}{35L^2} & \frac{7168}{35L^3} & 0 & \frac{-3584}{35L^3} & \frac{896}{35L^2} \\ \frac{1920}{35L^2} & \frac{320}{35L} & 0 & \frac{1280}{35L} & \frac{-1920}{35L^2} & \frac{320}{35L} \\ \frac{-1508}{35L^3} & \frac{-242}{35L^2} & \frac{-3584}{35L^3} & \frac{-1920}{35L^2} & \frac{5092}{35L^3} & \frac{-1138}{35L^2} \\ \frac{242}{35L^2} & \frac{38}{35L} & \frac{896}{35L^2} & \frac{320}{35L} & \frac{-1138}{35L^2} & \frac{332}{35L} \end{bmatrix} \begin{bmatrix} \upsilon_1 \\ \theta_1 \\ \upsilon_2 \\ \theta_2 \\ \upsilon_3 \\ \theta_3 \end{bmatrix} = \frac{L^2}{420}\begin{bmatrix} \frac{1}{L}(79q_1 + 19q_3) \\ (5q_1 + 2q_3) \\ \frac{112}{L}(q_1 + q_3) \\ (8q_3 - 8q_1) \\ \frac{1}{L}(19q_1 + 79q_3) \\ -(2q_1 + 5q_3) \end{bmatrix} \tag{24}$$

ou, simplesmente:

$$[k]\{u\} = \{f\} \tag{25}$$

sendo:

$[k]$: matriz de rigidez do elemento;

$\{u\}$:vetor de deslocamentos nodais do elemento;

$\{f\}$:vetor de forças e momentos nodais equivalentes.

5. ASPECTOS COMPUTACIONAIS

Neste trabalho foi desenvolvido um programa na linguagem JAVA para estudo de vigas, chamado de JVigas. No *software* foi implementado todo o equacionamento matemático descrito nesse trabalho. Como resultado das análises o *software* fornece valores, direções e sentidos das reações de apoio, configuração da viga deslocada e diagramas de força cortante e momento fletor.

É importante destacar que a entrada de dados e a saída dos resultados são feitas por meio de interface gráfica. A interface gráfica e todo o esquema geral de cálculo são descritos nos itens subsequentes deste trabalho.

5.1 APRESENTAÇÃO DA INTERFACE GRÁFICA

5.1.1 Janela inicial

Ao clicar sobre o ícone executável do programa JVigas, abre-se a janela inicial, conforme figura 5, que conta com uma barra de menu (1).

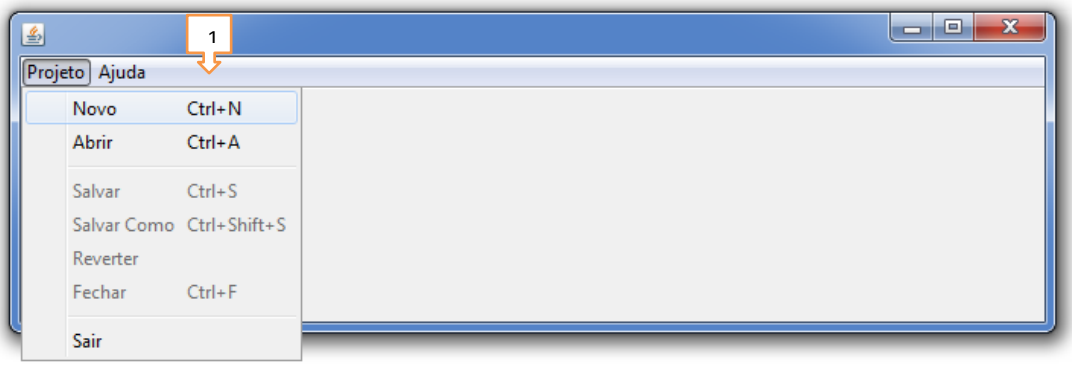


Figura 5: Janela de abertura do programa
Fonte: autoria própria.

5.1.2 Janela de um novo projeto

Clicando-se em “Projeto/Novo” surgem novos campos na janela (figura 6) conforme a descrição subsequente:

(1) espaço destinado para texto qualquer que sobre o projeto de viga;

(2) consta o endereço onde o arquivo foi salvo.

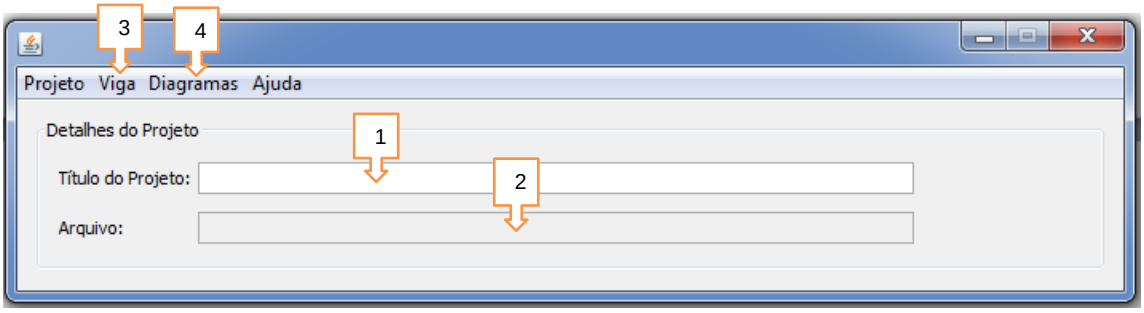


Figura 6: Janela de um novo projeto
Fonte: autoria própria.

No menu principal, o usuário, através da opção “Viga” (3), tem a acesso à:

- “Detalhes da viga”: permite a inclusão dos dados de entrada do projeto;
- “Dump de variáveis”: apresenta a matriz de rigidez global e o vetor de cargas global elaborados pelo sistema.

Ainda no menu principal, a opção “Diagramas” (4) dá acesso a:

- “Diagrama da estrutura da viga”: apresenta graficamente a viga e seus carregamentos e vinculações;
- “Reações de apoio”: ilustra o diagrama de corpo livre;
- “Diagrama de força cortante”: apresenta o diagrama de força cortante;
- “Diagrama de momento fletor”: apresenta o diagrama de momento fletor;
- “Deslocada da estrutura”: apresenta a viga em sua posição deslocada.

Na sequência é apresentada uma descrição mais completa das janelas mencionadas.

5.1.3 Janela “Detalhes da viga”

Acessando a opção “Detalhes da viga”, abre-se a janela para inserção das características de cada elemento da viga, conforme ilustra a figura 7. No campo (1) os dados referentes ao comprimento da viga, número total de elementos e número total de nós, são atualizados automaticamente a cada elemento gravado. A entrada de dados referentes aos elementos finitos segue as seguintes características:

- Campo (2): módulo de elasticidade longitudinal;
- Campo (3): momento de inércia em relação ao eixo principal;
- Campo (4): comprimento do elemento;

- Campo (5): carregamento distribuído sobre o elemento, conforme ilustra a figura 2;
- Campo (6): a opção “Giro” deverá ser marcada sempre que houver restrição ao giro no respectivo nó;
- Campo (7): a opção “Deslocamento em y” deverá ser marcada sempre que houver impedimento ao deslocamento vertical no nó em questão;
- Campo (8): na existência de um momento aplicado nos respectivos nós, este campo deve ser preenchido com seu valor;
- Campo (9): preencher com o valor da força concentrada nos respectivos nós.

As demais opções dessa janela são:

- Campo (10): grava o elemento inserido;
- Campo (11): diretório com a relação dos elementos adicionados;
- Campo (12): insere novos elementos ou remove elementos já existentes e divide pela metade um elemento selecionado mantendo suas características de rigidez.

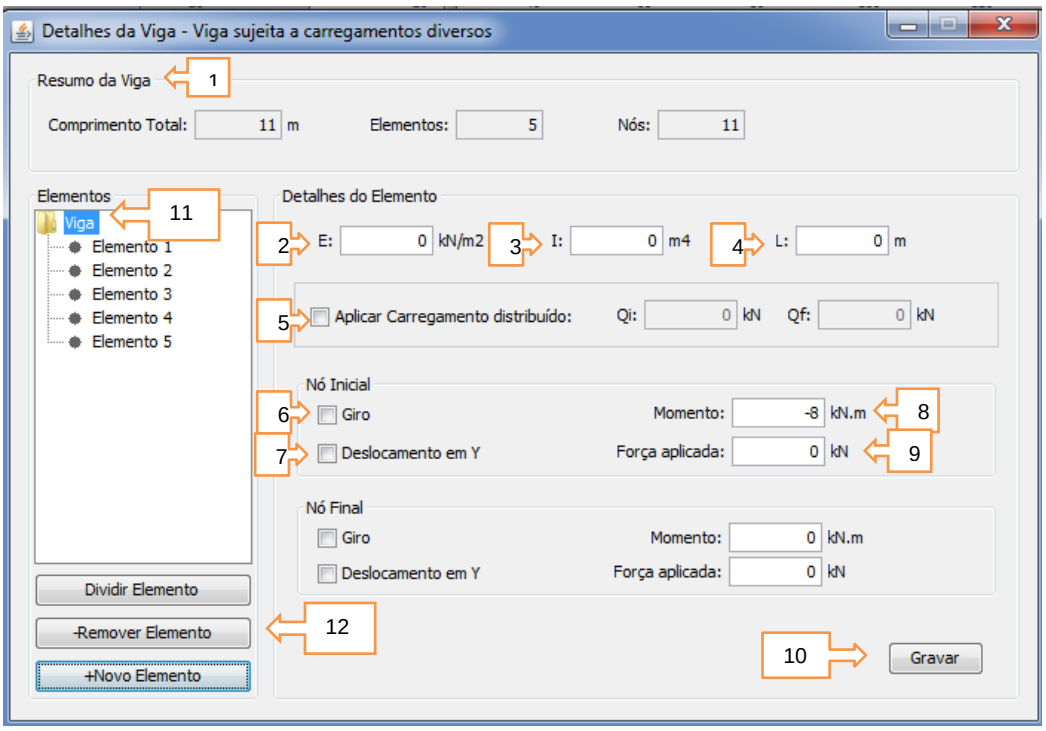


Figura 7: Janela “Detalhes da viga”
Fonte: Autoria própria.

É importante destacar que, após a inserção de um novo elemento, o *software* ilustra o modelo atualizado da viga (figura 8).

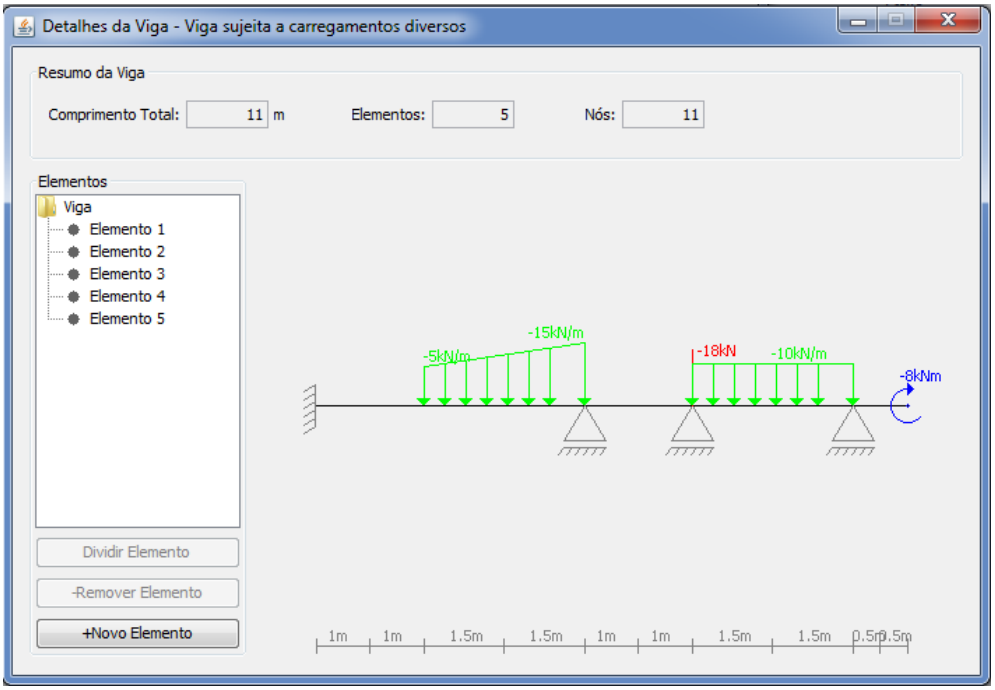


Figura 8: Resumo da viga
Fonte: Autoria própria.

5.1.4 Janela “Diagrama da estrutura da viga”

Apresenta a estrutura da viga, com seus carregamentos e vinculações, as linhas de cotas e, também botão com a opção para salvar a imagem do diagrama (figura 9).

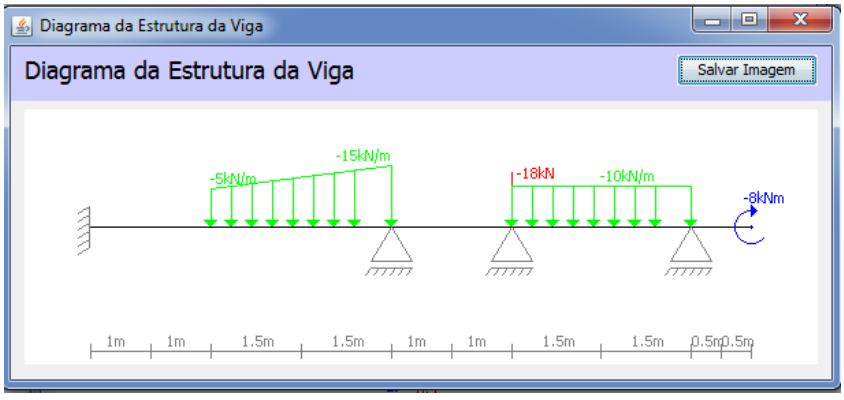


Figura 9: Janela “Diagrama da estrutura da viga”
Fonte: Autoria própria.

5.1.5 Janela “Reações de apoio”

Esta janela traz o diagrama de corpo livre da viga em análise, com opção para salvar a imagem do diagrama (Figura 10).

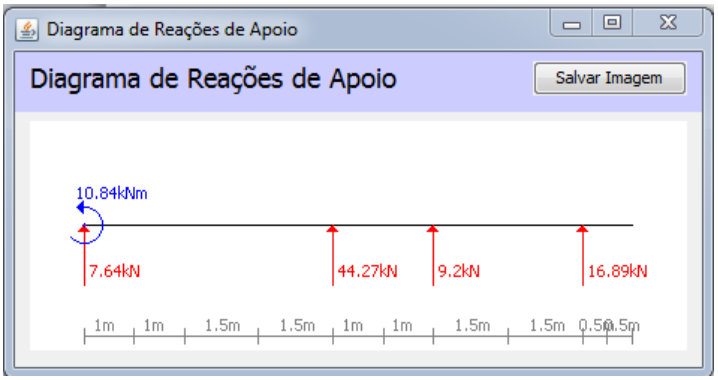


Figura 10: Janela “Reações de apoio”
Fonte: Autoria própria.

5.1.6 Janela “Diagrama de força cortante”

Ainda no sub-menu “Diagramas”, essa opção apresenta o diagrama de força cortante calculado com base na equação (30), apresentando os valores de cortante em todos os nós da estrutura (figura 11). O valor da força cortante pode ser obtido para qualquer ponto da viga, por meio de um clique com o botão esquerdo do *mouse* no ponto em que se deseja a informação. Após o clique, abre-se a janela “Inserir Régua” que vem com a coordenada do ponto em que foi clicado; esse valor pode ser alterado digitando-se a cota em metros do ponto da viga que desejar. Esse diagrama poderá ser gravado por meio do botão “Salvar Imagem”.

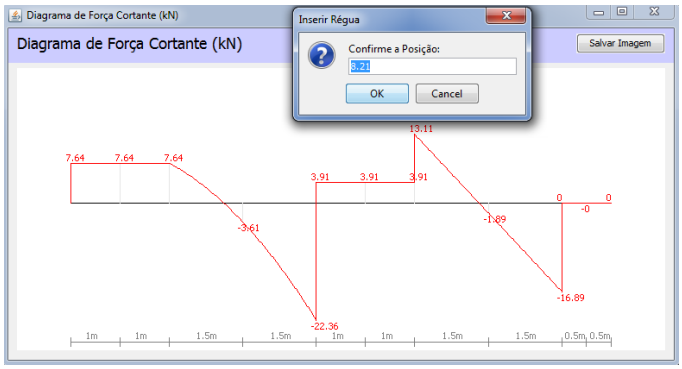


Figura 11: Janela Diagrama de Força Cortante
Fonte: Autoria própria.

5.1.7 Janela “Diagrama de momento fletor”

Ilustra o diagrama de momento fletor traçado com base na equação (28), constando os valores de momento fletor em todos os nós da estrutura (figura 12). Os valores do momento também poderão ser mostrados para qualquer ponto da viga através de um clique no ponto em que se deseja a informação, procedimento análogo ao descrito no item 5.1.6 e exemplificado na posição de 3,13m da figura 12.

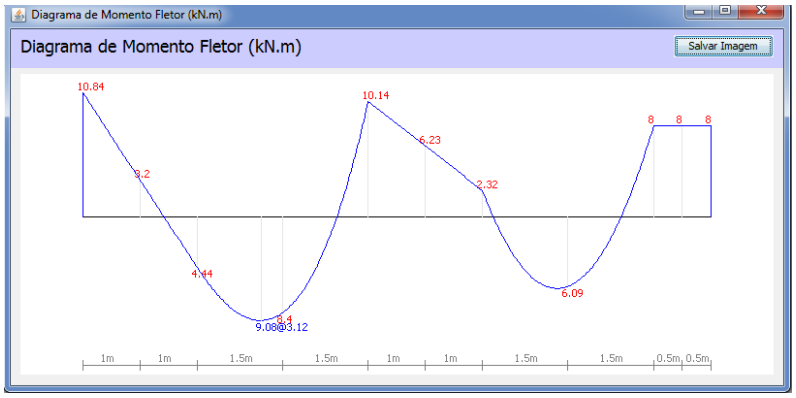


Figura 12: Janela “Diagrama de momento fletor”
Fonte: Autoria própria.

5.1.8 Janela “Deslocada da estrutura”

Apresenta, conforme ilustra a Figura 13 a posição deformada da estrutura, após aplicação dos carregamentos, com seu traçado determinado com base na equação (14). O valor do deslocamento em qualquer ponto da viga pode ser verificado clicando-se no ponto da em que se deseja a informação.

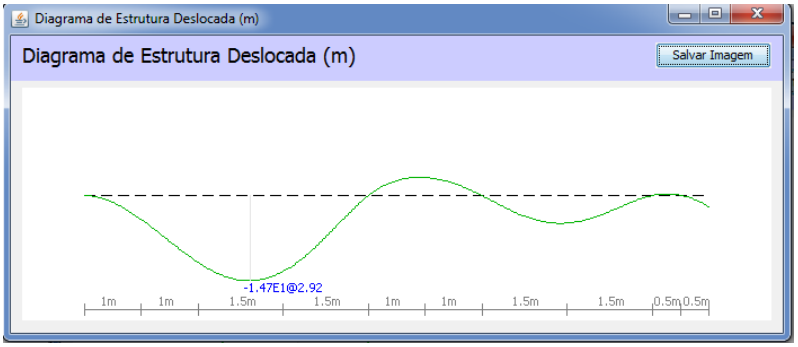


Figura 13: Janela “Deslocada da estrutura”
Fonte: Autoria própria.

5.2 ESQUEMA GERAL DE CÁLCULO

Todos os principais processos do algoritmo implementado serão comentados nas seções seguintes. De forma sucinta, e para melhor entendimento da sequência de cálculos, esses processos foram organizados na forma de fluxograma conforme ilustra a figura 13.

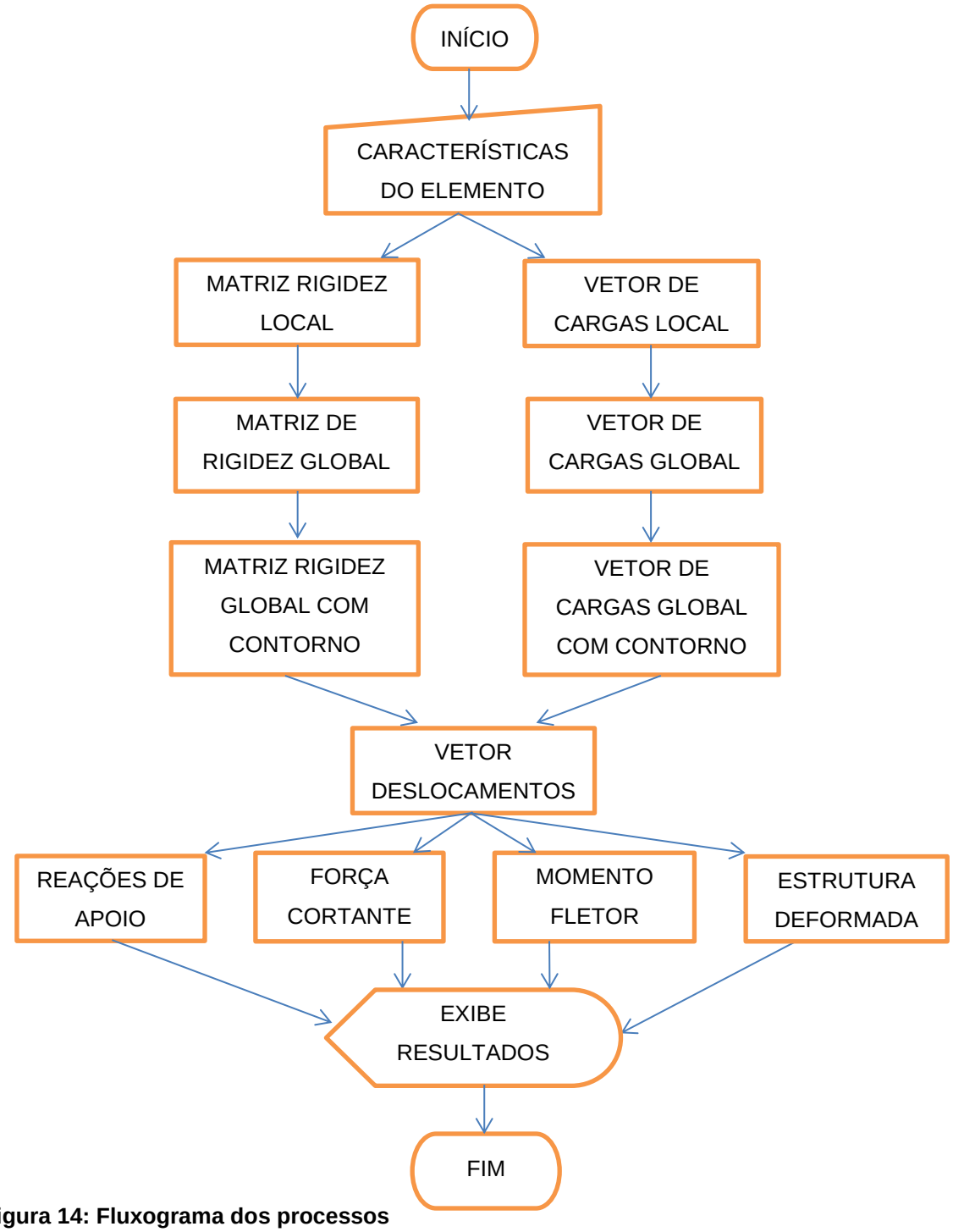


Figura 14: Fluxograma dos processos
Fonte: Autoria própria.

5.2.1 Matriz de rigidez local

Essa classe calcula a matriz de rigidez k de cada elemento, conforme a equação (19), a partir do seu módulo de elasticidade E, inércia I e comprimento L. Tais matrizes serão utilizadas para compor a matriz de rigidez global da estrutura.

5.2.2 Matriz de rigidez global

O algoritmo cria uma matriz quadrada de dimensões iguais ao número de graus de liberdade x número de nós da viga, para que nela sejam alocadas as matrizes de rigidez k dos elementos em suas posições corretas, formando assim a matriz de rigidez global K da estrutura. Nos nós que são comuns a dois elementos, há uma sobreposição de suas matrizes de rigidez e, portanto, esses valores devem ser somados para compor a matriz global. O procedimento, de maneira genérica, é ilustrado na equação (26).

$$K = \begin{bmatrix} k^1_{11} & k^1_{12} & k^1_{13} & k^1_{14} & k^1_{15} & k^1_{16} & 0 & \cdots & 0 \\ k^1_{21} & k^1_{22} & k^1_{23} & k^1_{24} & k^1_{25} & k^1_{26} & 0 & \cdots & 0 \\ k^1_{31} & k^1_{32} & k^1_{33} & k^1_{34} & k^1_{35} & k^1_{36} & 0 & \cdots & 0 \\ k^1_{41} & k^1_{42} & k^1_{43} & k^1_{44} & k^1_{45} & k^1_{46} & 0 & \cdots & 0 \\ k^1_{51} & k^1_{52} & k^1_{53} & k^1_{54} & k^1_{55}+k^2_{11} & k^1_{56}+k^2_{12} & k^2_{13} & \cdots & 0 \\ k^1_{61} & k^1_{62} & k^1_{63} & k^1_{64} & k^1_{65}+k^2_{21} & k^1_{66}+k^2_{22} & k^2_{23} & \cdots & 0 \\ 0 & 0 & 0 & 0 & k^2_{31} & k^2_{32} & k^2_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & k^n_{66} \end{bmatrix} \quad (26)$$

5.2.3 Vetor de cargas local

Nessa classe é calculado o vetor de cargas local, que é dado pela soma entre o vetor de forças nodais equivalentes (equação (23)), para o caso de existir carregamento distribuído ao longo do elemento, e o respectivo valor da força ou momento aplicado no nó.

5.2.4 Vetor de cargas global

Posiciona os vetores de cargas locais num vetor global referente a todos os elementos da viga, de modo que os valores relativos aos nós em comum entre elementos ocupem a mesma posição e seus valores sejam somados.

5.2.5 Condição de contorno

A matriz de rigidez global é uma matriz singular e, portanto, não inversível. Para permitir a resolução do sistema de equações e a consequente obtenção das variáveis nodais, é necessário introduzir previamente na matriz de rigidez as condições do contorno do problema.

O procedimento consiste em impor a condição de deslocamento ou giro nulo nos nós que possuam tais graus de liberdades restritos. Matematicamente, tal procedimento consiste em introduzir o número um na diagonal principal da matriz de rigidez referente a posição do nó/grau de liberdade restrito, zerando as demais posições da referente linha e coluna. Também deve-se zerar a posição correspondente no vetor de cargas global. Esse tratamento pode ser visualizado, de maneira genérica, através da equação (27).

$$K = \begin{bmatrix} k^1_{11} & 0 & k^1_{13} & k^1_{14} & 0 & k^1_{16} & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ k^1_{31} & 0 & k^1_{33} & k^1_{34} & 0 & k^1_{36} & 0 & \cdots & 0 \\ k^1_{41} & 0 & k^1_{43} & k^1_{44} & 0 & k^1_{46} & 0 & \cdots & 0 \\ k^1_{51} & 0 & k^1_{53} & k^1_{54} & 0 & k^1_{56} + k^2_{12} & k^2_{13} & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & k^2_{32} & k^2_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & k^n_{66} \end{bmatrix} \qquad F = \begin{bmatrix} F^1_1 \\ 0 \\ F^1_2 \\ M^1_2 \\ F^1_3 \\ 0 \\ F^2_2 \\ \vdots \\ M^n_3 \end{bmatrix} \qquad (27)$$

5.2.6 Deslocamento

O vetor deslocamento é obtido a partir da solução do sistema linear de equações. Neste trabalho, a resolução do sistema se dá pelo método de eliminação de Gauss com pivoteamento parcial.

5.2.7 Reações de apoio

Nessa classe o algoritmo contempla o cálculo do vetor reações de apoio através da multiplicação entre a matriz de rigidez global e o vetor deslocamentos.

Vale ressaltar que o vetor obtido apresenta todos os valores de reações bem como os valores das forças nodais provenientes do carregamento aplicado. Assim, para a determinação correta das reações de apoio, o vetor obtido deve ser subtraído do vetor de cargas global.

5.2.8 Momento fletor

Essa classe do algoritmo contempla o cálculo dos momentos fletores em qualquer ponto do elemento finito. Considerando a equação diferencial da linha elástica, os momentos fletores podem ser interpolados a partir das variáveis nodais como segue:

$$M(x) = EI\left[\frac{d^2\phi_1}{dx^2}\upsilon_1 + \frac{d^2\phi_2}{dx^2}\theta_1 \frac{d^2\phi_3}{dx^2}\upsilon_2 + \frac{d^2\phi_4}{dx^2}\theta_2 + \frac{d^2\phi_5}{dx^2}\upsilon_3 + \frac{d^2\phi_6}{dx^2}\theta_3\right] \tag{28}$$

sendo:

$$\begin{aligned}
\frac{d^2\phi_1}{dx^2} &= \frac{2}{L^5}\left(240x^3 - 408Lx^2 + 198L^2x - 23L^3\right)\\
\frac{d^2\phi_2}{dx^2} &= \frac{2}{L^4}\left(40x^3 - 72Lx^2 + 39L^2x - 6L^3\right)\\
\frac{d^2\phi_3}{dx^2} &= \frac{32}{L^4}\left(6x^2 - 6Lx + L^2\right)\\
\frac{d^2\phi_4}{dx^2} &= \frac{16}{L^4}\left(2x - L\right)\left(10x^2 - 10Lx + L^2\right)\\
\frac{d^2\phi_5}{dx^2} &= \frac{2}{L^5}\left(-240x^3 + 312Lx^2 - 102L^2x + 7L^3\right)\\
\frac{d^2\phi_6}{dx^2} &= \frac{2}{L^4}\left(40x^3 - 48Lx^2 + 15L^2x - L^3\right)
\end{aligned} \tag{29}$$

No caso de nós que são comuns a dois elementos, dois valores de momentos fletores nodais são determinados, sendo o cálculo baseado nas variáveis nodais de cada elemento adjacente. Tal procedimento permite a obtenção dos saltos nos diagramas de momento fletor para o caso de momentos concentrados.

Vale ressaltar que, pela convenção de giros adotada anteriormente, o momento fletor será positivo quando o giro da seção for anti-horário.

5.2.9 Força cortante

De maneira análoga ao cálculo do momento fletor, essa classe permite a obtenção do valor de força cortante em qualquer ponto do elemento. Novamente, em casos de nós comuns, dois valores de força cortante são obtidos visando a detecção de saltos no diagrama de força cortante.

A partir da relação diferencial entre força cortante e momento fletor, os valores de força cortante podem ser obtidos como segue:

$$V(x) = EI\left[\frac{d^3\phi_1}{dx^3}\upsilon_1 + \frac{d^3\phi_2}{dx^3}\theta_1 \frac{d^3\phi_3}{dx^3}\upsilon_2 + \frac{d^3\phi_4}{dx^3}\theta_2 + \frac{d^3\phi_5}{dx^3}\upsilon_3 + \frac{d^3\phi_6}{dx^3}\theta_3\right] \qquad (30)$$

sendo:

$$
\begin{aligned}
\frac{d^3\phi_1}{dx^3} &= \frac{2}{L^5}\left(720x^2 - 816Lx + 198L^2\right) \\
\frac{d^3\phi_2}{dx^3} &= \frac{2}{L^4}\left(120x^2 - 144Lx + 39L^2\right) \\
\frac{d^3\phi_3}{dx^3} &= \frac{32}{L^4}\left(12x - 6L\right) \\
\frac{d^3\phi_4}{dx^3} &= \frac{16}{L^4}\left(60x^2 - 60Lx + 12L^2\right) \\
\frac{d^3\phi_5}{dx^3} &= \frac{2}{L^5}\left(-720x^2 + 624Lx - 102L^2\right) \\
\frac{d^3\phi_6}{dx^3} &= \frac{2}{L^4}\left(120x^2 - 96Lx + 15L^2\right)
\end{aligned}
\tag{31}
$$

5.2.10 Deslocada da estrutura

Nessa classe é determinado o valor de deslocamento em qualquer ponto do elemento. Os valores são interpolados a partir das variáveis nodais conforme descrito na equação (14).

5.2.11 Exibe resultados

Essa classe é responsável por elaborar os diagramas que podem ser acessados através do menu principal.

6. RESULTADOS E DISCUSSÕES

6.1 EXEMPLO 1

Este exemplo aborda o estudo de uma viga bi apoiada com comprimento igual a 4 metros e submetida a um carregamento uniformemente distribuído igual a 10 kN/m. A viga tem ainda rigidez à flexão $EI = 10.000\ kN.m^2$ constante em todo o seu comprimento.

A figura 15, ilustra a configuração geométrica, condições de vinculação e carregamento consideradas no exemplo. Ressalta-se que as figuras apresentadas foram retiradas diretamente do JVigas.

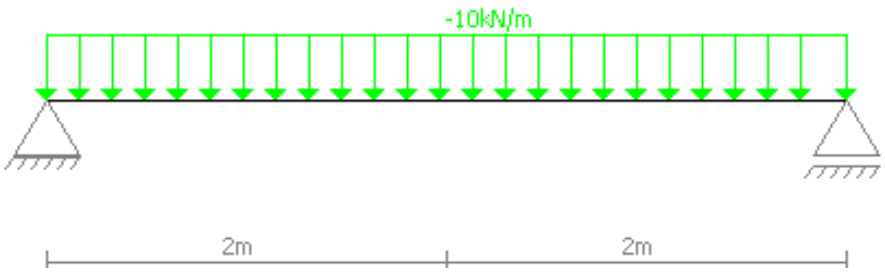


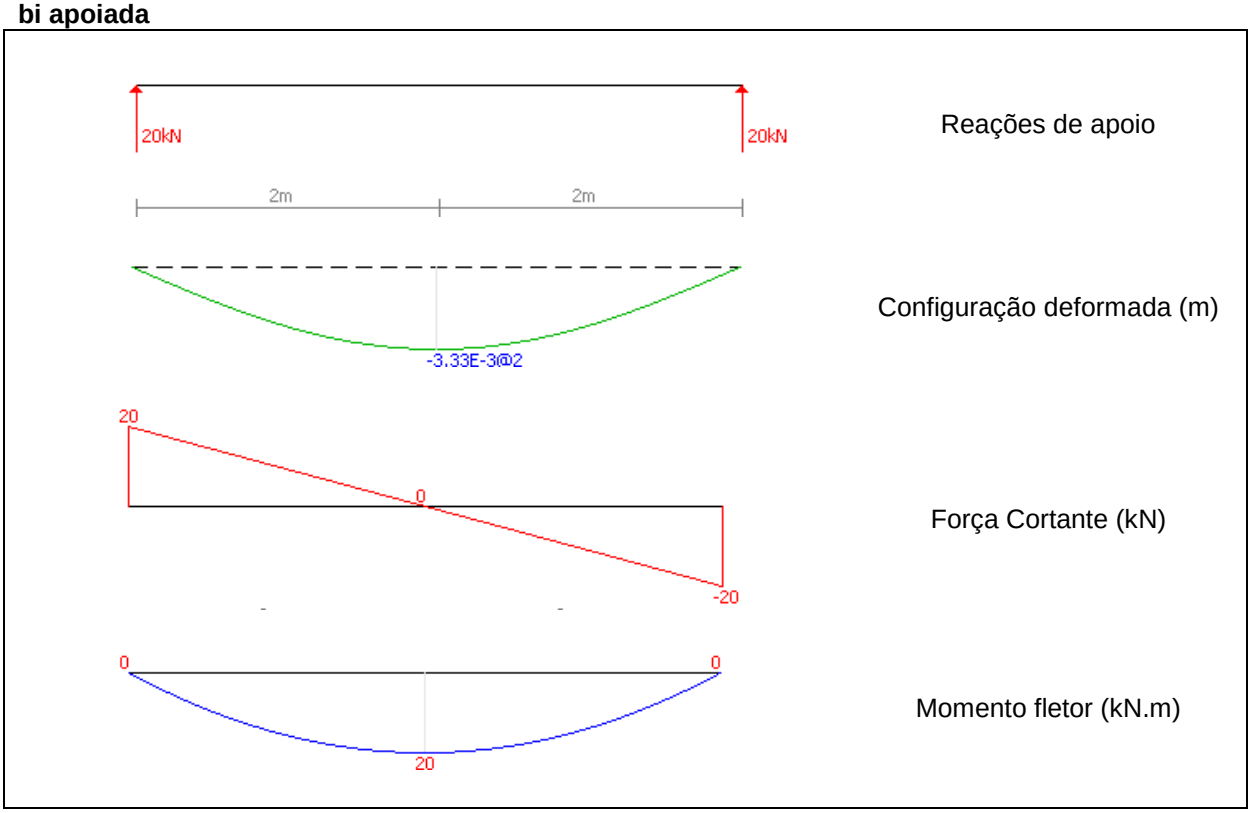
Figura 15: Estrutura da viga
Fonte: Autoria própria - JVigas.

Para a simulação numérica do exemplo em questão foi utilizada uma malha com um único elemento finito. Como resultados foram obtidos os diagramas de corpo livre, força cortante e momento fletor, bem como a viga em sua configuração deslocada. Os diagramas estão ilustrados no quadro 1.

Este exemplo foi proposto inicialmente por Castro (2009) que utilizou em suas análises elementos finitos com função de aproximação do terceiro grau. Em seu trabalho, o autor discute a perda de precisão nos resultados de esforços solicitantes em decorrência da diminuição do grau das funções de aproximação à medida em que são avaliados a partir das derivadas da função deslocamento. Esse procedimento é o mesmo adotado neste trabalho, conforme abordado nos itens 6.2.8 e 6.2.9.

Assim, de modo a realizar uma análise comparativa, as figurasFigura 16,Figura 17 e Figura 18 trazem os resultados fornecidos pelo autor juntamente com os exatos e os obtidos por meio do JVigas.

Quadro 1: Estrutura deslocada, reações de apoio, força cortante e momento fletor para a viga bi apoiada



Fonte: Autoria própria – JVigas

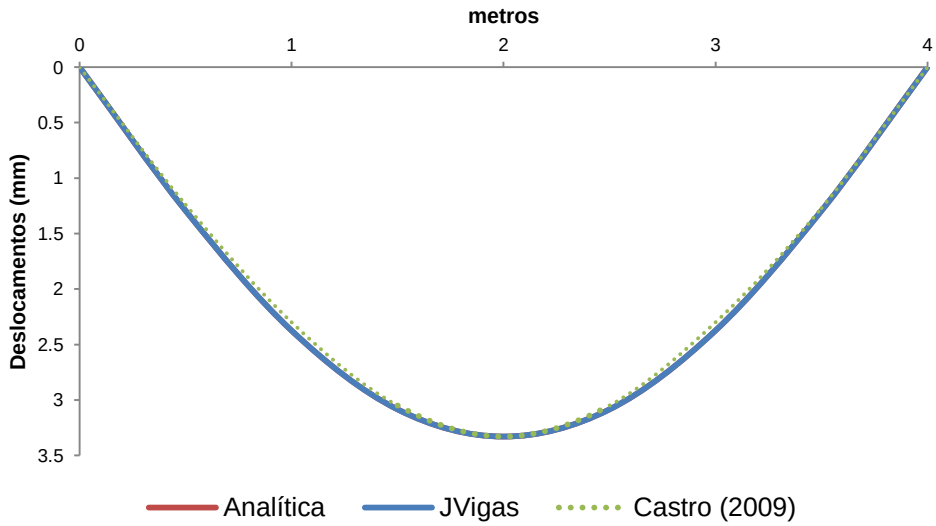


Figura 16: Viga bi apoiada - Configuração deslocada da estrutura
Fonte: Adaptado de Castro (2009, p. 54).

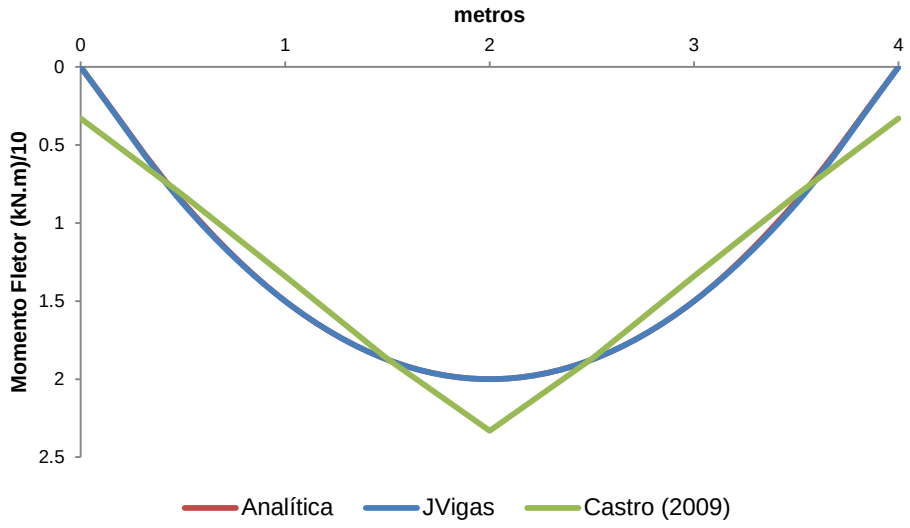


Figura 17: Viga bi apoiada - Diagramas de momentos fletores (kN.m)
Fonte: Adaptado de Castro (2009, p. 54).

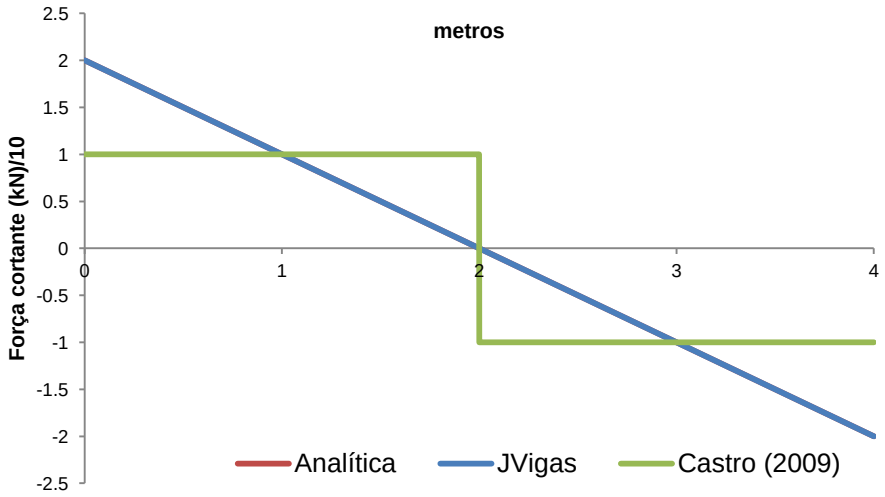


Figura 18: Viga bi apoiada - Diagramas de forças cortantes (kN)
Fonte: Adaptado de Castro (2009, p. 54).

A Figura 16 traz uma comparação entre as soluções obtidas utilizando-se elementos finitos com diferentes graus de aproximação com a solução obtida através da equação da linha elástica. Em ambos os casos as soluções se comportam muito bem, praticamente coincidindo com a solução analítica.

A medida em que se compara os modelos em termos de resultados de momento fletor (Figura 17) o que se percebe é que a solução fornecida por Castro (2009) tende a se distanciar da solução analítica, inclusive no que se refere ao grau da função representativa do momento fletor ao longo da viga. Essa perda de precisão demonstrada é originada em função da utilização, conforme demonstrado

no item 5.2.8 deste trabalho, da equação diferencial da linha elástica para a avaliação do momento fletor ao longo do elemento.

Para o problema abordado neste exemplo, o carregamento uniformemente distribuído implica em equação de linha elástica e diagrama de momento fletor representadas por polinômios de quarto e segundo graus, respectivamente. Castro (2009) utilizou em suas análises um modelo baseado na discretização através de elementos finitos com função de aproximação do terceiro grau, implicando em momentos fletores representados por funções lineares. O modelo, então, apresenta perda de precisão da solução a medida em que se aumenta o grau da função representativa do carregamento.

O mesmo fato pode ser verificado em termos de resultados de força cortante. O modelo proposto Castro (2009) tende a fornecer soluções ainda menos precisas, pois a solução para esse tipo de esforço solicitante é obtida através de funções constantes para interpolar as variáveis nodais.

No que se refere ao modelo implementado neste trabalho, o grau polinômio interpolador do elemento finito tende a coincidir com o polinômio representativo da solução analítica. Assim, a solução tende a se apresentar de forma exata para problemas envolvendo carregamentos distribuídos lineares ou de grau inferior. Esse fato pode ser comprovado nos resultados ilustrados nas figuras já referenciadas.

Como alternativa à essa questão, Castro (2009) propõe a utilização de malhas mais refinadas, o que implica em um maior custo computacional, ou a soma da equação da linha elástica de uma viga bi engastada na função interpoladora do elemento finito.

Vale ressaltar que em ambos os trabalhos os valores de força cortante e momento fletor nodais não foram obtidos pelo sistema local $k \cdot u = f$ e que não foram utilizadas técnicas de pós processamento.

6.2 EXEMPLO 2

Um segundo exemplo proposto por Castro (2009) é abordado neste item. Trata-se de uma viga contínua de 8 metros de comprimento, sujeita a ação de uma força concentrada igual a $10\ kN$ e a um carregamento uniformemente distribuído

com valor igual a 5 kN/m (figura 19). A viga tem rigidez à flexão EI constante em todo o seu comprimento de 8 m.

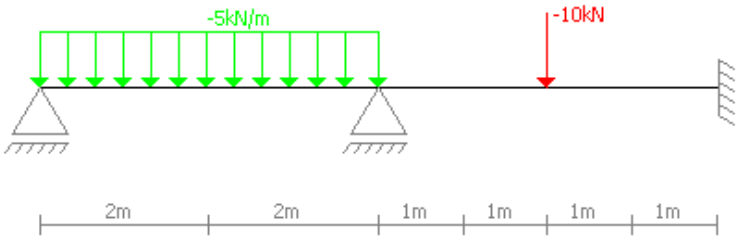


Figura 19: Viga contínua sujeita a carregamentos diversos
Fonte: Autoria própria – JVigas.

Castro (2009) utilizou em suas análises uma malha composta por 11 elementos fintos. Já na simulação realizada através do *software* JVigas utilizou-se uma malha composta por 3 elementos dispostos conforme ilustra a figura 20.

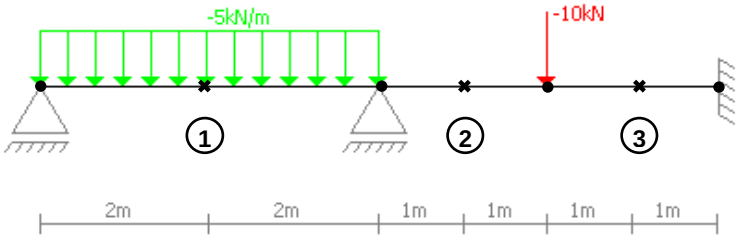
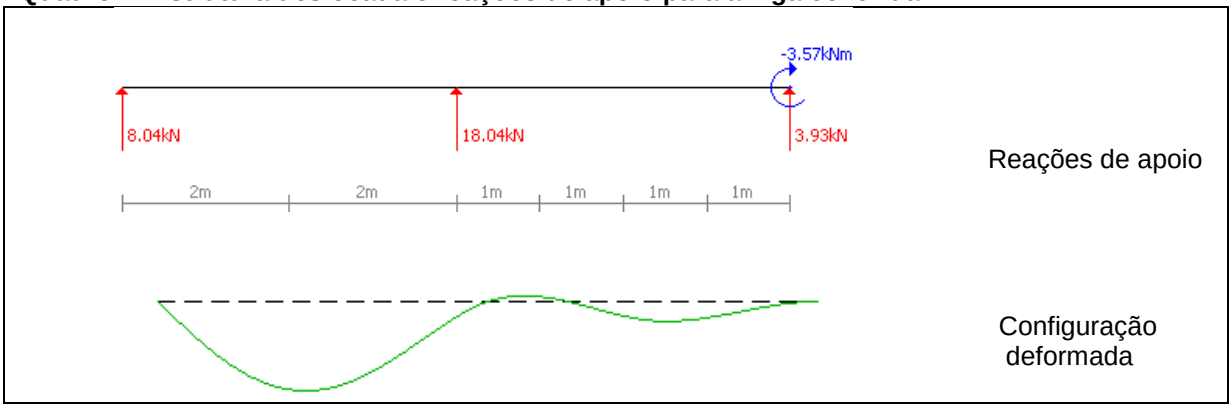


Figura 20: Viga contínua – Malha utilizada
Fonte: Autoria própria – JVigas.

O quadro 2 apresenta o diagrama de corpo livre e a viga em sua configuração deslocada obtidos a partir do *software* implementado. Já as figuras 21 e 22 apresentam um comparativo em termos de momentos fletores e forças cortantes entre as soluções fornecidas pelo *software* JVigas, por Castro (2009) e pela solução analítica.

Quadro 2: Estrutura deslocada e reações de apoio para a viga contínua



Fonte: Autoria própria – JVigas

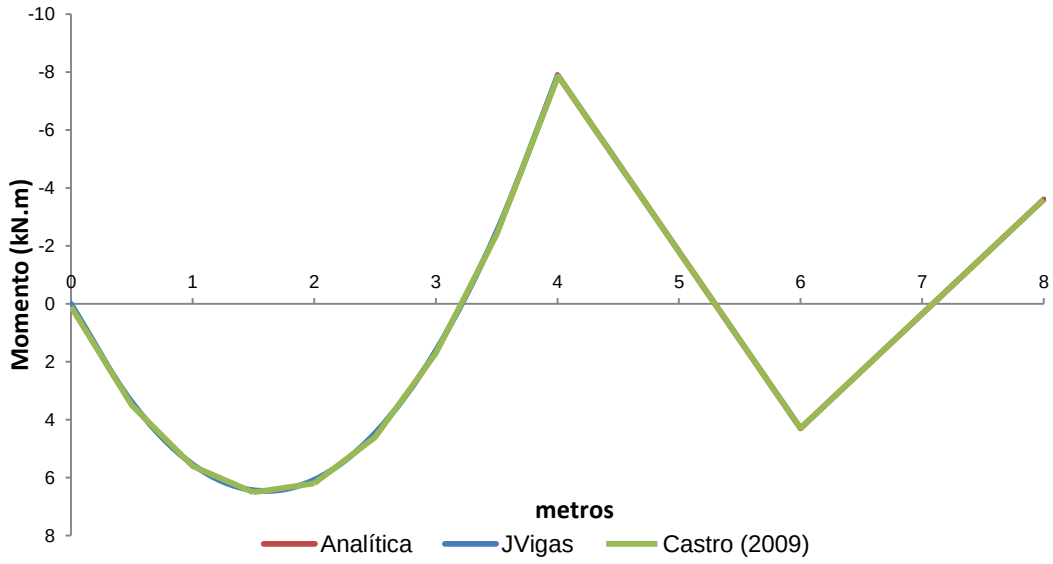


Figura 21: Viga contínua – Diagramas de momentos fletores (kN.m)
Fonte: Adaptado de Castro (2009, p. 54).

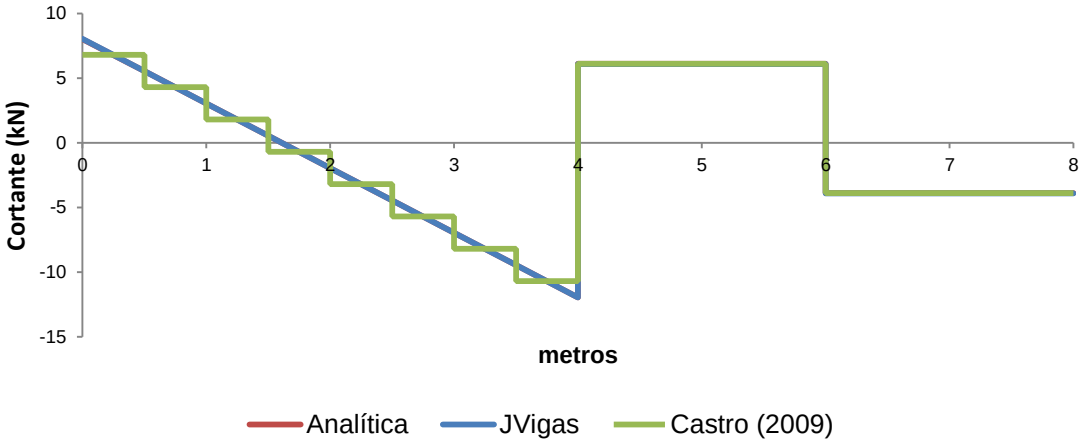


Figura 22: Viga bi apoiada - Diagramas de forças cortantes (kN)
Fonte: Adaptado de Castro (2009, p. 54).

Diferentemente do caso abordado no exemplo 1, este problema aborda duas situações distintas em termos de carregamento. O primeiro tramo da viga contínua está submetida a um carregamento uniforme enquanto que o segundo está submetido a uma força concentrada. Em termos de linha elástica, o primeiro tramo requer para a sua definição um polinômio do quarto grau, enquanto que o segundo é definido através de um polinômio cúbico.

Através das figuras 21 e 22 pode-se notar que, para o procedimento de solução apresentado, as soluções obtidas através do *software* JVigas por Castro (2009) coincidem com a solução analítica em elementos que estão submetidos

apenas a forças concentradas. Isso se explica pelo fato de a função de aproximação utilizada pelo autor coincidir com o grau exato e a formulação aqui apresentada utilizar de um polinômio de grau superior ao da função necessária para a definição dos deslocamentos ao longo do elemento.

A situação se altera na medida em que se trabalha com carregamentos distribuídos. Nestes casos, a formulação aqui apresentada tende a apresentar resultados exatos em simulações envolvendo carregamentos uniformes ou linearmente distribuídos ao longo do elemento, ao passo que a apresentada pelo autor apresenta soluções aproximadas.

Por fim o exemplo ilustra também a necessidade de refinamento de malha em situações onde se deseja melhorar a qualidade da solução almejada. A medida em que se utiliza um número maior de elementos finitos, a solução obtida tende a se aproximar cada vez mais da solução analítica. Já para o caso onde se utiliza de elementos de grau de aproximação mais elevado, a solução coincide com a solução analítica com a utilização de malhas mínimas e, assim, com menor custo computacional.

6.3 EXEMPLO 3

Este terceiro exemplo aborda o caso de uma viga contínua, uma vez hiperestática, de seção transversal retangular de dimensões iguais a 14 x 50 cm $\left(I \cong 1{,}46 \times 10^5 \text{ mm}^4\right)$. A viga é feita de concreto armado C 25 $\left(E = 23800 \text{ MPa}\right)$ e está sujeita aos carregamentos ilustrados na Figura 23.

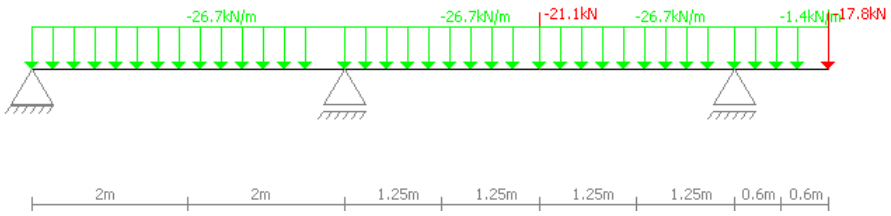
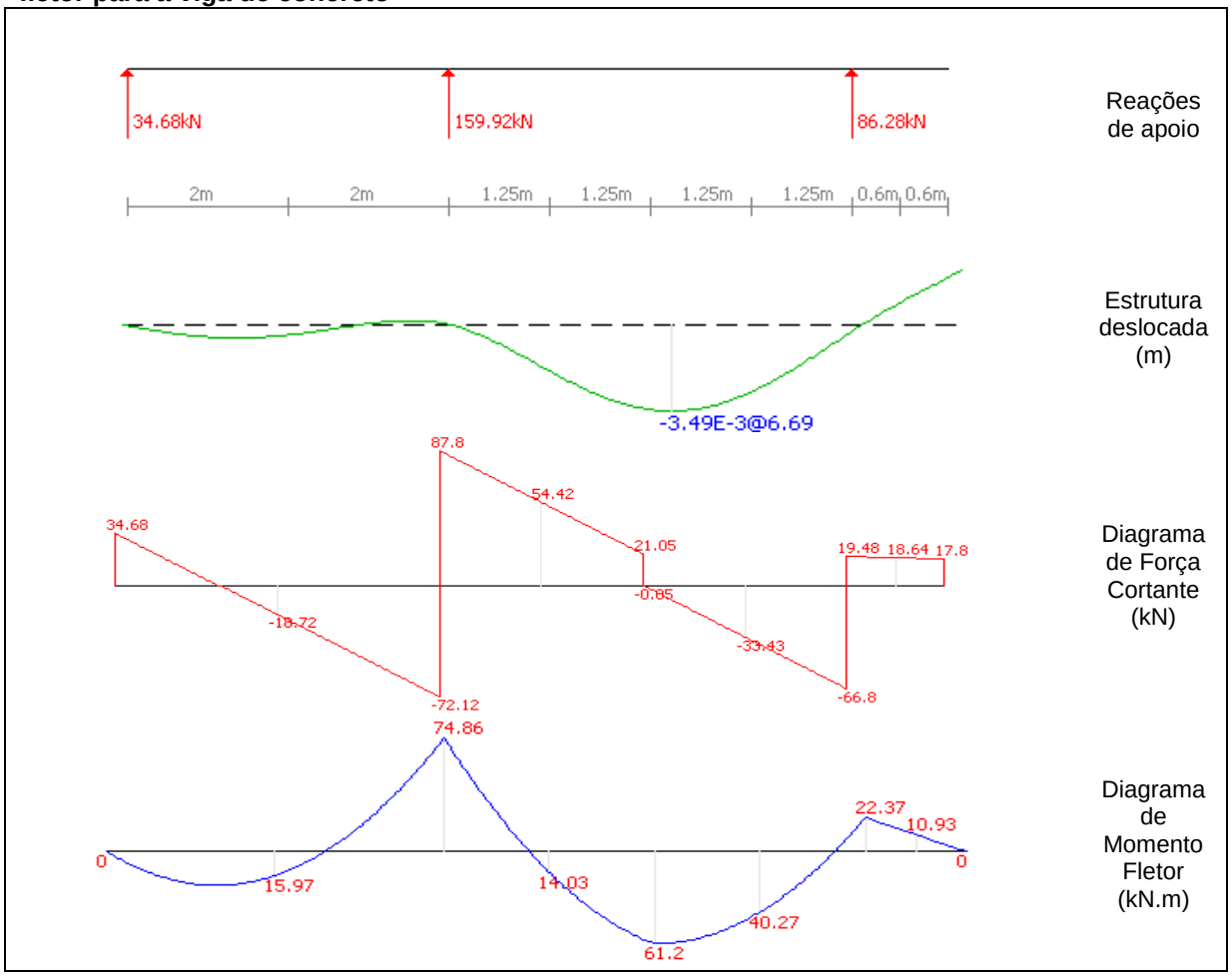


Figura 23: Viga contínua uma vez hiperestática, sujeita a carregamentos diversos
Fonte: Autoria própria - JVigas

A malha utilizada na análise é composta de três elementos finitos, sendo um em cada vão e um na extremidade em balanço. Como resultados da simulação

numérica, o programa JVigas apresentou os resultados representados nos diagramas ilustrados no quadro 2.

Quadro 2: Estrutura deslocada, reações de apoio, diagramas de força cortante e momento fletor para a viga de concreto



Fonte: Autoria própria – JVigas.

Buscando validar os resultados obtidos, a mesma simulação foi realizada utilizando o *software* Ftool. O diagrama gerado para o momento fletor é ilustrado na Figura 24 para comparação dos resultados.

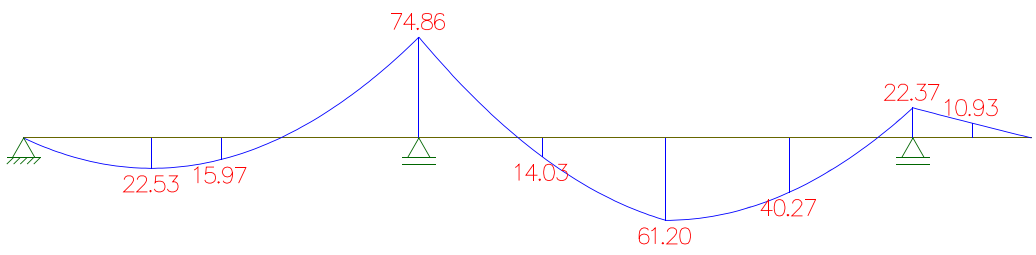


Figura 24: Diagrama de momento fletor (kN.m)
Fonte: Ftool.

A análise comparativa dos diagramas de momento fletor apresentados permite concluir que ambas as análises forneceram os mesmos resultados. A mesma observação pode ser feita a partir de análises dos diagramas de força cortante e de deslocamentos. Apenas para evitar repetições, esses últimos resultados obtidos a partir do programa Ftool foram suprimidos deste trabalho.

O exemplo analisado vem novamente ilustrar a precisão do algoritmo proposto.

6.4 EXEMPLO 4

Este último exemplo aborda o caso de uma viga 6 vezes hiperestática, com tramos de diferentes rigidezes à flexão, sendo $EI = 2000\ kN \cdot m^2$, e sujeita a ação de diversas cargas conforme ilustra a fFigura 25.

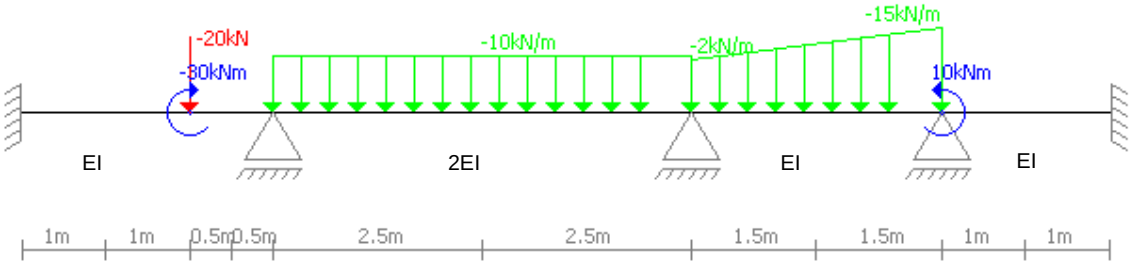


Figura 25: Viga 6 vezes hiperestática sujeita a carregamentos diversos
Fonte: Autoria própria – Jvigas.

A simulação numérica do problema no *software* implementando foi realizada utilizando-se uma malha composta por 5 elementos finitos (Figura 26). O quadro 4 apresenta os diagramas de estado e também a viga em sua configuração deformada obtidos a partir do *software* JVigas. Para efeitos de comparação, a Figura 27 traz o diagrama de momento fletor para a viga obtido a partir da simulação no *software* Ftool.

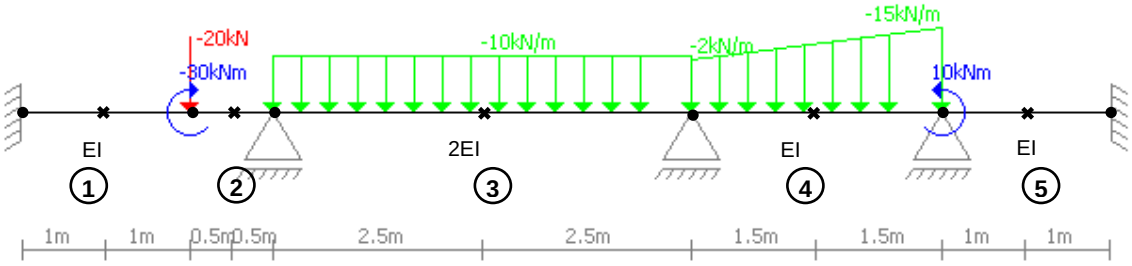
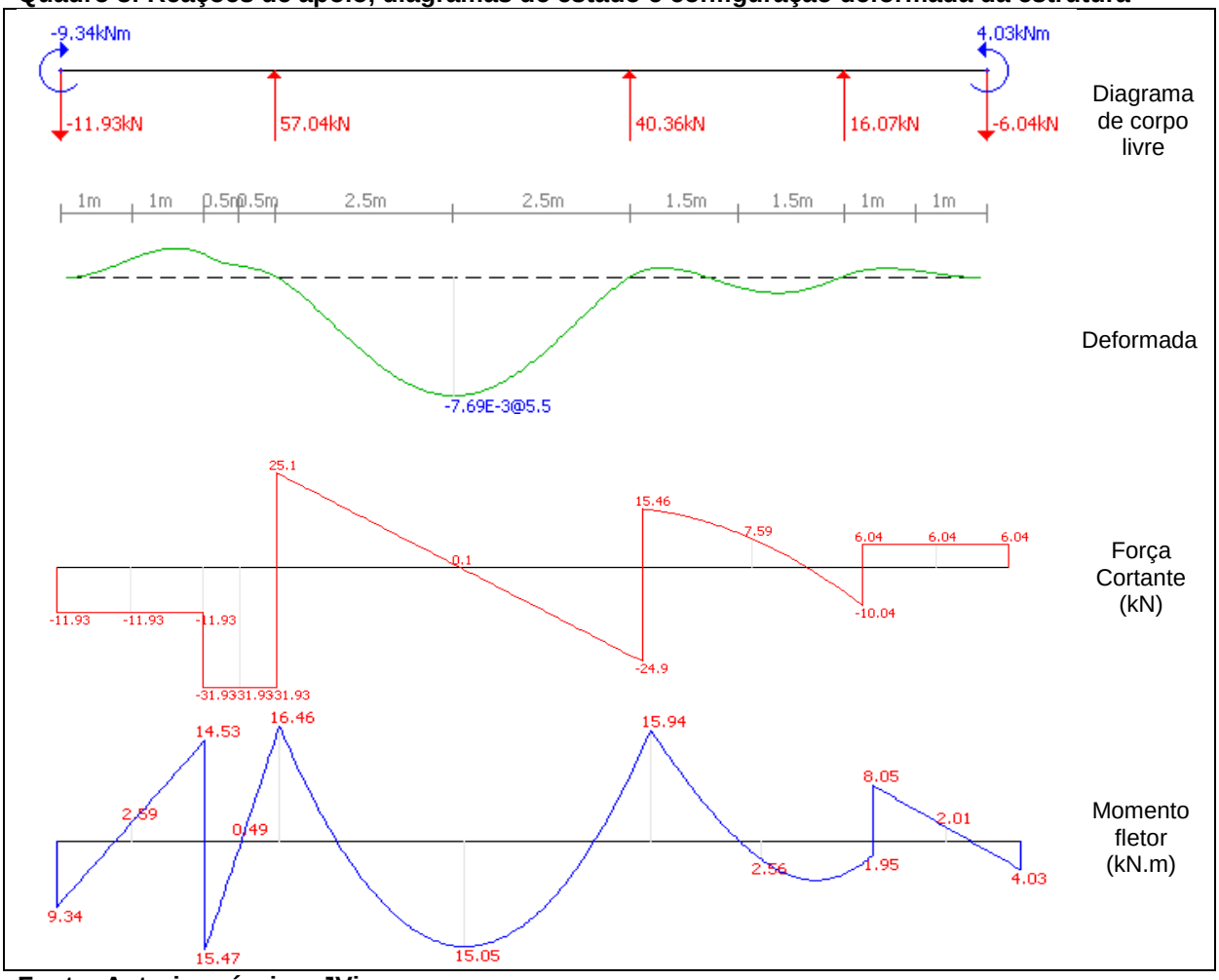


Figura 26: Malha utilizada na análise
Fonte: Autoria própria – JVigas.

Quadro 3: Reações de apoio, diagramas de estado e configuração deformada da estrutura



Fonte: Autoria própria – JVigas.

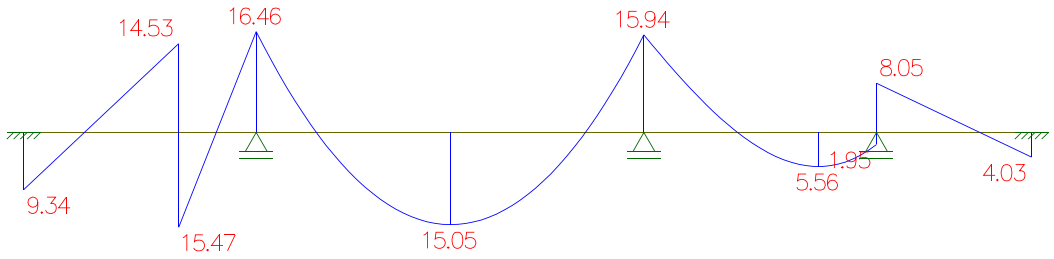


Figura 27: Diagrama de momento fletor (kN.m)
Fonte: Ftool.

O objetivo deste exemplo é apresentar o comportamento do algoritmo na análise de vigas sujeitas a carregamentos linearmente distribuídos, momentos concentrados e diferentes rigidezes ao longo de seu comprimento.

A análise comparativa em termos de momento fletor indica que o *software* JVigas forneceu soluções precisas a um baixo custo computacional. A exemplo do ocorrido no exemplo anterior deste trabalho, a análise em termos de deslocamentos e força cortante também mantém a precisão, sendo que os diagramas fornecidos pelo *software* Ftool foram também suprimidos do trabalho.

7. CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo apresentar um algoritmo fundamentado no MEF para análise linear de vigas de Euler-Bernoulli, utilizando particularmente elementos finitos lineares com polinômios de quinto grau, para aproximar o campo de deslocamentos ao longo do seu comprimento.

Para o desenvolvimento matemático necessário, inicialmente partiu-se do Princípio dos Trabalhos Virtuais e demais conceitos da mecânica dos sólidos deformáveis, de modo a se obter a matriz de rigidez e o vetor de cargas do elemento. A sequência do trabalho se deu pelo desenvolvimento do algoritmo capaz de montar a matriz de rigidez e o vetor de cargas global da estrutura, resolver o sistema de equações lineares para a obtenção das incógnitas nodais (deslocamentos e giros) e, por fim, determinar as reações de apoio, a força cortante e o momento fletor ao longo de cada elemento. É importante ressaltar que neste trabalho o cálculo das forças internas nos elementos finitos é feito a partir da equação diferencial da linha elástica e pela relação diferencial entre força cortante e momento fletor.

O algoritmo elaborado foi então implementado computacionalmente, em linguagem Java, e deu origem ao *software* JVigas. Como principal característica, o *software* apresenta a interface gráfica capaz de ilustrar a viga em sua configuração deformada, seu diagrama de corpo livre, diagramas de força cortante e momento fletor ao longo de seu comprimento.

De forma a validar o *software* elaborado, quatro exemplos foram apresentados ao longo do trabalho. Os resultados das simulações foram comparados com as soluções analíticas, as fornecidas por Castro (2009) e também com as obtidas pelo *software* Ftool e ilustraram a obtenção de resultados precisos para problemas que envolvam carregamentos concentrados, uniformes e linearmente distribuídos.

Por fim, é importante ressaltar que em todas as simulações foram utilizadas malhas reduzidas, indicando que o algoritmo é capaz de oferecer soluções precisas a baixo custo computacional e sem a necessidade de se utilizar de técnicas de pós processamento.

Sendo assim, todos os objetivos propostos por nesse trabalho foram alcançados.

REFERÊNCIAS

ALVES FILHO, Avelino. **Elementos finitos: a base da tecnologia CAE**. 1. ed. São Pauo, SP: Érica, 2000.

AZEVEDO, Álvaro F. M. **Método dos elementos finitos**. 1. ed. Porto, 2003. 248 p. Disponível em: <http://www.fe.up.pt/~alvaro>. Acesso em: 03 fev. 2014.

BEER, Ferdinand P.; JOHNSTON JR, E. Russell. **Resistência dos materiais**. 3. ed. São Paulo: Makron, 1996.

BEER, Ferdinand Pierre; JOHNSTON, E. Russel. **Mecânica vetorial para engenheiros**. 5. ed. São Paulo, SP: Pearson Makron Books, 1994.

BRAZ, José Felipe. **Análise estrutural de uma base guia submarina pelo método dos elementos finitos**. 2011. 99 f. Dissertação – Departamento de Engenharia Mecânica da Universidade de Taubaté, Taubaté, 2011. Disponível em: <http://www.bdtd.unitau.br/tedesimplificado/tde_busca/arquivo.php?codArquivo=297>. Acesso em: 28 jul. 2014.

CAMPILHO, Raul Duarte Salgueiral Gomes. **Método de elementos finitos – ferramentas para análise estrutural**. 1. ed. Porto, Portugal: PUBLINDUSTRIA, 2012. Disponível em: <issuu.com/engebook/docs/mef-paginas>. Acesso em: 28 jul. 2014.

CASTRO, Luís Manuel Santos. **Método dos elementos finitos: análise de pórticos planos**. 2009. 130 f. Notas de Aula – IST Instituto Técnico de Lisboa, Portugal. Disponível em: <http://www.civil.ist.utl.pt/~luis/ae2/AE2_Portico.pdf>. Acesso em: 01 ago. 2014.

FISH, Jacob; BELYTSCHKO, Ted. **Um primeiro curso em elementos finitos**. 1. ed. Rio de Janeiro, RJ: LTC, 2009.

HIBBELER, R.C.. **Resistência dos materiais**. 3. ed. Rio de Janeiro, RJ: LTC, 2000.

NASCIMENTO, Rangel Ferreira do. **Análise dinâmica de vigas utilizando o elemento finito de Timoshenko com refinamento P-adaptativo**. 2005. 212 f. Dissertação – Faculdade de Engenharia de Ilha solteira da Universidade Estadual Paulista Júlio Mesquita Filho, Ilha Solteira, 2005. Disponível em: <http://www.feis.unesp.br/Home/departamentos/engenhariamecanica/pos-graduacao/dissertacao_rfnascimento.pdf>. Acesso em: 16 jun. 2014.

POPOV, Egor P. **Introdução à mecânica dos sólidos**. São Paulo, SP: E. Blücher, 1978.

RAQUEL, S. Lotti; MACHADO, André W.; MAZZIEIRO, Ênio T.; LANDRE, Janes. **Aplicabilidade científica do método dos elementos finitos**. R Dental Press Ortodon Ortop. Facial. Maringá, v. 11, n. 2, p. 35-43, mar/abril 2006. Disponível em: <http://www.scielo.br/pdf/dpress/v11n2/a06v11n2.pdf>. Acesso em: 28 jul. 2014.

RIBEIRO, José Carlos Lopes. **Simulação via método dos elementos finitos da disitribuição tridimensional de temperatura em estruturas em situação de incêndio**. 2004. 178 f. Dissertação – Escola de Engenharia da Universidade Federal de Minas Gerais, Belo Horizonte, 2004. Disponível em: <http://www.bibliotecadigital.ufmg.br/dspace/handle/1843/FACO-6AYG27>. Acesso em: 28 jul. 2014.

SORIANO, Humberto Lima. **Análise de estruturas formulação matricial e implementação computacional**. 1. ed. Rio de Janeiro, RJ: Editora Ciência Moderna, 2009.

SORIANO, Humberto Lima. **Elementos finitos – formulação e aplicação na estática e dinâmica das estruturas**. 1. ed. São Paulo, SP: Érica, 2000.

SOUZA, Remo Magalhães de. **O método dos elementos finitos aplicado ao problema de condução de calor**. Belém, 2003. Disponível em: <http://www.inf.ufes.br/~luciac/fem/livros-fem/ApostilaElementosFinitosNiCAE.pdf>. Acesso em: 28 jul. 2014.

TIMOSHENKO, Stephen. **Mecânica dos sólidos**. 1 ed. Rio de Janeiro, RJ: LTC 1984.

UGURAL, Ansel C. **Mecânica dos materiais**. 1. ed. Rio de Janeiro, RJ: LTC, 2009.

APÊNDICE A – Código fonte dos principais processos do JVigas

```
public class Elemento {

    /* informações coletadas do elemento */
    private double E;
    private double I;
    private double L;
    private boolean W; // carregamento distribuído
    private double Qi;
    private double Qf;
    private No noInicial;
    private No noInter;
    private No noFinal;

    /* informações calculadas do elemento */
    private int numeroElemento;
    private final MatrizRigidezLocal matrizRigidez;
    private final CargasLocal cargas;

    /* */
    Elemento anterior;
    Elemento proximo;

    public Elemento(double E, double I, double L, boolean W, double Qi, double Qf, No noInicial, No
noFinal, No noInter) {
        this.E = E;
        this.I = I;
        this.L = L;
        this.W = W;
        this.Qi = Qi;
        this.Qf = Qf;
        this.noInicial = noInicial;
        this.noFinal = noFinal;
        this.noInter = noInter;

        this.matrizRigidez = new MatrizRigidezLocal(this);
        this.cargas = new CargasLocal(this);

        // numero do elemento na viga
        this.numeroElemento = 0;
    }

    void setX(double x) {
        // atualiza coordenada x dos Nós
        noInicial.setX(x);
        noFinal.setX(x + this.L);
        noInter.setX(x + this.L / 2); // nó intermediário
    }

    void setNumeroElemento(int n) {
        if (n > 0) {
            int desloca = 2 * (n - 1);
            this.noInicial.setNumeroNo(1 + desloca);
            this.noInter.setNumeroNo(2 + desloca);
            this.noFinal.setNumeroNo(3 + desloca);

            this.numeroElemento = n;
        } else {
```

```
            this.numeroElemento = 0;
        }
    }

    public double getE() {
        return E;
    }

    public double getI() {
        return I;
    }

    public double getL() {
        return L;
    }

    public boolean isW() {
        return W;
    }

    public double getQi() {
        return Qi;
    }

    public double getQf() {
        return Qf;
    }

    public No getNoInicial() {
        return noInicial;
    }

    public No getNoInter() {
        return noInter;
    }

    public No getNoFinal() {
        return noFinal;
    }

    public double getX() {
        return noInicial.getX();
    }

    public Elemento getAnterior() {
        return anterior;
    }

    public Elemento getProximo() {
        return proximo;
    }

    public void setE(double E) {
        this.E = E;
    }

    public void setI(double I) {
        this.I = I;
    }
```

```
    public void setL(double L) {
        this.L = L;
    }

    public void setW(boolean W) {
        this.W = W;
    }

    public void setQi(double Qi) {
        this.Qi = Qi;
    }

    public void setQf(double Qf) {
        this.Qf = Qf;
    }

    public void setNoInicial(No noInicial) {
        this.noInicial = noInicial;
    }

    public void setNoFinal(No noFinal) {
        this.noFinal = noFinal;
    }

    public int getNumeroElemento() {
        return numeroElemento;
    }

    public MatrizRigidezLocal getMatrizRigidez() {
        return matrizRigidez;
    }

    public CargasLocal getCargas() {
        return cargas;
    }
}
public class No {
    /* informações do Nó */
    private double M; // momento
    private double P; // carregamento aplicado
    private boolean giro;
    private boolean deslocamentoY;

    /* informacoes calculads */
    private double x;
    private int numeroNo;

    public No(double M, double P, boolean giro, boolean deslocamentoY) {
        this.M = M;
        this.P = P;
        this.giro = giro;
        this.deslocamentoY = deslocamentoY;

        this.numeroNo = 0;
    }

    void copiar(No no) {
        this.M = no.M;
        this.P = no.P;
        this.deslocamentoY = no.deslocamentoY;
```

```
        this.giro = no.giro;
        this.numeroNo = no.numeroNo;
        this.x = no.x;
    }

    void setX(double x) {
        this.x = x;
    }

    void setNumeroNo( int n ) {
        this.numeroNo = n;
    }

    public double getX() {
        return x;
    }

    public double getM() {
        return M;
    }

    public double getP() {
        return P;
    }

    public boolean isGiro() {
        return giro;
    }

    public boolean isDeslocamentoY() {
        return deslocamentoY;
    }

    public void setM(double M) {
        this.M = M;
    }

    public void setP(double P) {
        this.P = P;
    }

    public void setGiro(boolean giro) {
        this.giro = giro;
    }

    public void setDeslocamentoY(boolean deslocamentoY) {
        this.deslocamentoY = deslocamentoY;
    }

    public int getNumeroNo() {
        return numeroNo;
    }
}
public class CargasLocal {

    private final Elemento elemento;

    public CargasLocal(Elemento elemento) {
        this.elemento = elemento;
    }
```

```
public double[] getVetorAplicadas() {
    // forças dos nós
    double ip = elemento.getNoInicial().getP();
    double im = elemento.getNoInicial().getM();

    double mp = elemento.getNoInter().getP();
    double mm = elemento.getNoInter().getM();

    double fp = elemento.getNoFinal().getP();
    double fm = elemento.getNoFinal().getM();

    double vetCargasPlicadas[] = new double[6];
    vetCargasPlicadas[0] = ip;
    vetCargasPlicadas[1] = im;
    vetCargasPlicadas[2] = mp;
    vetCargasPlicadas[3] = mm;
    vetCargasPlicadas[4] = fp;
    vetCargasPlicadas[5] = fm;

    return vetCargasPlicadas;
}

public double[] getVetorDistribuidas() {
    double parcCargDist[] = new double[6];

    if (elemento.isW()) {
        double L = elemento.getL();
        double q1 = elemento.getQi();
        double q3 = elemento.getQf();

        double mult = Math.pow(L, 2) / 420.0;

        parcCargDist[0] = mult * (79 * q1 + 19 * q3) / L;
        parcCargDist[1] = mult * (5 * q1 + 2 * q3);
        parcCargDist[2] = mult * 112 * (q1 + q3) / L;
        parcCargDist[3] = mult * (8 * q3 - 8 * q1);
        parcCargDist[4] = mult * (19 * q1 + 79 * q3) / L;
        parcCargDist[5] = mult * -1 * (2 * q1 + 5 * q3);
    } else {
        for (int i = 0; i < 6; i++) {
            parcCargDist[i] = 0.0;
        }
    }

    return parcCargDist;
}

public double[] getVetor() {
    double cargasTotais[] = new double[6];

    double vetAplicadas[] = getVetorAplicadas();
    double vetDistribuidas[] = getVetorDistribuidas();

    for(int i=0; i<cargasTotais.length; i++){
        cargasTotais[i] = vetAplicadas[i] + vetDistribuidas[i];
    }

    return cargasTotais;
}
```

```
}

public class CargasGlobal extends VigaAtributo {

    private double vetorCargas[];

    public CargasGlobal(Viga viga) {
        super(viga);
        this.vetorCargas = null;
    }

    public double[] getVetor() {
        verificarAtualizacoesViga();

        return vetorCargas;
    }

    @Override
    protected void calcular() {
        // matriz zerada
        double vetCargasGlobal[];
        int tamanho = 2 * viga.getQtdNos();

        // vetor zerado
        vetCargasGlobal = new double[tamanho];
        double vetCargasGlobal_distribuidas[] = new double[tamanho];
        for (int i = 0; i < tamanho; i++) {
            vetCargasGlobal_distribuidas[i] = 0.0;
        }

        Elemento el = viga.getElementoInicial();
        while (el != null) {
            double vetCargasLocal[];

            int numPrimeiroNo = el.getNoInicial().getNumeroNo();
            int deslocX = 2 * numPrimeiroNo - 1 - 1;

            // copia forças aplicadas
            vetCargasLocal = el.getCargas().getVetorAplicadas();
            for (int i = 0; i < 6; i++) {
                vetCargasGlobal[i + deslocX] = vetCargasLocal[i];
            }

            // acumula forças distribuidas
            vetCargasLocal = el.getCargas().getVetorDistribuidas();
            for (int i = 0; i < 6; i++) {
                vetCargasGlobal_distribuidas[i + deslocX] += vetCargasLocal[i];
            }

            el = el.getProximo();
        }

        // acumula forças distribuídas nas aplicadas
        for(int i=0; i<tamanho; i++ ) {
            vetCargasGlobal[i] += vetCargasGlobal_distribuidas[i];
        }

        this.vetorCargas = vetCargasGlobal;
    }
}
```

```
public class MatrizRigidezLocal {

    private final double matConstantes[][];

    private final Elemento elemento;

    public MatrizRigidezLocal(Elemento elemento) {
        this.elemento = elemento;

        // define constantes
        matConstantes = new double[6][6];

        matConstantes[0][0] = 5092.0 / 35;
        matConstantes[0][1] = 1138.0 / 35;
        matConstantes[0][2] = -3584.0 / 35;
        matConstantes[0][3] = 1920.0 / 35;
        matConstantes[0][4] = -1508.0 / 35;
        matConstantes[0][5] = 242.0 / 35;

        matConstantes[1][0] = 1138.0 / 35;
        matConstantes[1][1] = 332.0 / 35;
        matConstantes[1][2] = -896.0 / 35;
        matConstantes[1][3] = 320.0 / 35;
        matConstantes[1][4] = -242.0 / 35;
        matConstantes[1][5] = 38.0 / 35;

        matConstantes[2][0] = -3584.0 / 35;
        matConstantes[2][1] = -896.0 / 35;
        matConstantes[2][2] = 7168.0 / 35;
        matConstantes[2][3] = 0.0 / 35;
        matConstantes[2][4] = -3584.0 / 35;
        matConstantes[2][5] = 896.0 / 35;

        matConstantes[3][0] = 1920.0 / 35;
        matConstantes[3][1] = 320.0 / 35;
        matConstantes[3][2] = 0.0 / 35;
        matConstantes[3][3] = 1280.0 / 35;
        matConstantes[3][4] = -1920.0 / 35;
        matConstantes[3][5] = 320.0 / 35;

        matConstantes[4][0] = -1508.0 / 35;
        matConstantes[4][1] = -242.0 / 35;
        matConstantes[4][2] = -3584.0 / 35;
        matConstantes[4][3] = -1920.0 / 35;
        matConstantes[4][4] = 5092.0 / 35;
        matConstantes[4][5] = -1138.0 / 35;

        matConstantes[5][0] = 242.0 / 35;
        matConstantes[5][1] = 38.0 / 35;
        matConstantes[5][2] = 896.0 / 35;
        matConstantes[5][3] = 320.0 / 35;
        matConstantes[5][4] = -1138.0 / 35;
        matConstantes[5][5] = 332.0 / 35;
        // - - -

    }

    private double getConstantePara(int lin, int col) {
        return matConstantes[lin][col];
```

```
    }

    private int getExpoentePara(int lin, int col) {
        if (lin % 2 == 0) {
            if (col % 2 == 0) {
                return 3;
            } else {
                return 2;
            }
        } else {
            if (col % 2 == 0) {
                return 2;
            } else {
                return 1;
            }
        }
    }

    // calcula
    public double[][] getMatriz() {

        // matriz zerada
        double matRigidez[][] = new double[6][6];
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 6; j++) {
                matRigidez[i][j] = 0.0;
            }
        }

        double L = elemento.getL();
        double E = elemento.getE();
        double I = elemento.getI();

        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 6; j++) {
                double numerador = getConstantePara(i, j);
                int expoente = getExpoentePara(i, j);

                double val = E * I * numerador / Math.pow(L, expoente);

                matRigidez[i][j] = val;
            }
        }

        return matRigidez;
    }
}

public class MatrizRigidezGlobal extends VigaAtributo {
    private double[][] matrizRigidez;

    public MatrizRigidezGlobal(Viga viga) {
        super(viga);

        this.matrizRigidez = null;
    }

    public double[][] getMatriz() {
        verificarAtualizacoesViga();
```

```
        return matrizRigidez;
    }

    @Override
    protected void calcular() {

        // matriz zerada
        double matRigidez[][];
        int tamanho = 2 * viga.getQtdNos();
        matRigidez = new double[tamanho][tamanho];
        for (int i = 0; i < tamanho; i++) {
            matRigidez[i] = new double[tamanho];
            for (int j = 0; j < tamanho; j++) {
                matRigidez[i][j] = 0.0;
            }
        }

        Elemento el = viga.getElementoInicial();
        while (el != null) {
            int numPrimeiroNo = el.getNoInicial().getNumeroNo();

            int deslocX = 2 * numPrimeiroNo - 1 - 1;
            int deslocY = 2 * numPrimeiroNo - 1 - 1;

            double[][] matRigidLocal = el.getMatrizRigidez().getMatriz();

            for (int i = 0; i < 6; i++) {
                for (int j = 0; j < 6; j++) {
                    matRigidez[i + deslocX][j + deslocY] += matRigidLocal[i][j];
                }
            }

            el = el.getProximo();
        }

        this.matrizRigidez = matRigidez;
    }
}

public class SolucaoContorno extends VigaAtributo {

    private double[] vetCargasContorno;
    private double[][] matRigidezContorno;

    public SolucaoContorno(Viga viga) {
        super(viga);

        this.vetCargasContorno = null;
        this.matRigidezContorno = null;
    }

    public double[] getVetorCargas() {
        verificarAtualizacoesViga();

        return vetCargasContorno;
    }


    public double[][] getMatrizRigidez() {
        verificarAtualizacoesViga();
```

```
    return matRigidezContorno;
}


// calcula condicao de contorno
@Override
protected void calcular() {
    double[][] matRigidez = viga.getMatrizRigidez().getMatriz();
    double[] vetCargas = viga.getCargas().getVetor();
    int n = vetCargas.length;

    vetCargasContorno = new double[n];
    matRigidezContorno = new double[n][n];

    // copia matriz e vetor
    System.arraycopy(vetCargas, 0, vetCargasContorno, 0, n);
    for (int i = 0; i < n; i++) {
        matRigidezContorno[i] = new double[n];
        System.arraycopy(matRigidez[i], 0, matRigidezContorno[i], 0, n);
    }

    Elemento el = viga.getElementoInicial();

    while (el != null) {
        int desloc = 2 * el.getNoInicial().getNumeroNo() - 1 - 1;

        if (el.getNoInicial().isDeslocamentoY()) {
            int d = 0 + desloc;
            zerarMatrizLinhaColuna(matRigidezContorno, d);
            vetCargasContorno[d] = 0.0; // zera também vetor de cargas
            matRigidezContorno[d][d] = 1;
        }
        if (el.getNoInicial().isGiro()) {
            int d = 1 + desloc;
            zerarMatrizLinhaColuna(matRigidezContorno, d);
            vetCargasContorno[d] = 0.0; // zera também vetor de cargas
            matRigidezContorno[d][d] = 1;
        }

        if (el.getNoInter().isDeslocamentoY()) {
            int d = 2 + desloc;
            zerarMatrizLinhaColuna(matRigidezContorno, d);
            vetCargasContorno[d] = 0.0; // zera também vetor de cargas
            matRigidezContorno[d][d] = 1;
        }
        if (el.getNoInter().isGiro()) {
            int d = 3 + desloc;
            zerarMatrizLinhaColuna(matRigidezContorno, d);
            vetCargasContorno[d] = 0.0; // zera também vetor de cargas
            matRigidezContorno[d][d] = 1;
        }

        if (el.getNoFinal().isDeslocamentoY()) {
            int d = 4 + desloc;
            zerarMatrizLinhaColuna(matRigidezContorno, d);
            vetCargasContorno[d] = 0.0; // zera também vetor de cargas
            matRigidezContorno[d][d] = 1;
        }
        if (el.getNoFinal().isGiro()) {
```

```
                int d = 5 + desloc;
                zerarMatrizLinhaColuna(matRigidezContorno, d);
                vetCargasContorno[d] = 0.0; // zera também vetor de cargas
                matRigidezContorno[d][d] = 1;
            }

            el = el.getProximo();
        }
    }

    private void zerarMatrizLinhaColuna(double[][] mat, int n) {
        // zera dados da coluna
        for (int z = 0; z < mat.length; z++) {
            mat[z][n] = 0;
        }

        // zera dados da linha
        for (int z = 0; z < mat[n].length; z++) {
            mat[n][z] = 0;
        }
    }

}
public class Deslocamentos extends VigaAtributo {

    double vetorDeslocamentos[];

    private MatrizNaoInversivelException ex = null;


    public Deslocamentos(Viga viga) {
        super(viga);

        this.vetorDeslocamentos = null;
    }


    public double[] getVetor() throws MatrizNaoInversivelException {
        verificarAtualizacoesViga();

        if ( ex != null) {
            throw ex;
        }

        return vetorDeslocamentos;
    }


Resolucao_Sistema(A,X,B,n)
    @Override
    protected void calcular() {
        this.ex = null; // zera marcador de erros de calculo


        int n;
        double[][] A, Triang, Pivo;
        double[] X, B;
        double Max;
        double Aux;
        double Soma;
```

```
int linha;

// inicializa variáveis
A = viga.getSolucaoContorno().getMatrizRigidez();
B = viga.getSolucaoContorno().getVetorCargas();

n = B.length;

X = new double[n];
for (int i = 0; i < n; i++) {
    X[i] = 0.0;
}

Triang = new double[n][n + 1];
for (int i = 0; i < n; i++) {
    Triang[i] = new double[n + 1];
}

Pivo = new double[n][n];
for (int i = 0; i < n; i++) {
    Pivo[i] = new double[n];
}

//Modificando a matriz A
for (int ii = 0; ii < n; ii++) {
    for (int jj = 0; jj < n; jj++) {
        Triang[ii][jj] = A[ii][jj];
    }
}

for (int ii = 0; ii < n; ii++) {
    Triang[ii][n + 1 - 1] = B[ii];
}

linha = -1; // Nunca usa este valor. Inicialização forçada pelo compilador.

// Construção do sistema equivalente triangular superior
for (int kk = 0; kk < n - 1; kk++) {
    Max = 0.0;
    for (int ii = kk; ii < n; ii++) {
        if (Math.abs(A[ii][kk]) > Math.abs(Max)) {
            Max = A[ii][kk];
            linha = ii;
        }
    }

    if (Max == 0) {
        // Erro no cálculo: Matriz não inversível. Impossível continuar.
        this.vetorDeslocamentos = null;
        this.ex = new MatrizNaoInversivelException();
        return;
    }

    for (int jj = kk; jj < n + 1; jj++) {
        Aux = Triang[linha][jj];
        Triang[linha][jj] = Triang[kk][jj];
        Triang[kk][jj] = Aux;
    }

    for (int ii = kk + 1; ii < n; ii++) {
```

```
                Pivo[ii][kk] = Triang[ii][kk] / Triang[kk][kk];

                for (int jj = kk; jj < n + 1; jj++) {
                    Triang[ii][jj] = Triang[ii][jj] - Pivo[ii][kk] * Triang[kk][jj];
                }
            }

            // Slução do sistema triangular superior
            X[n - 1] = Triang[n - 1][n + 1 - 1] / Triang[n - 1][n - 1];
            for (int ii = n - 1 - 1; ii >= 0; ii--) {
                Soma = 0.0;

                for (int jj = ii + 1; jj < n; jj++) {
                    Soma += Triang[ii][jj] * X[jj];
                }

                X[ii] = (Triang[ii][n + 1 - 1] - Soma) / Triang[ii][ii];
            }
        }


        this.vetorDeslocamentos = X;
    }

}
public class Deslocada extends VigaAtributo {

    public Deslocada(Viga viga) {
        super(viga);
    }

    @Override
    protected void calcular() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated
methods, choose Tools | Templates.
    }

    public double getDeslocadaAt(Elemento elemento, double x) throws MatrizNaoInversivelException
{
        return calcDeslocada(elemento, x, viga.getDeslocamentos().getVetor());
    }


    private double calcDeslocada(Elemento elemento, double x, double deslocamentos[]) {
        double L = elemento.getL();
        double L2 = Math.pow(L, 2);
        double L3 = Math.pow(L, 3);
        double L4 = Math.pow(L, 4);
        double L5 = Math.pow(L, 5);

        double X = x;
        double X2 = Math.pow(X, 2);
        double X3 = Math.pow(X, 3);
        double X4 = Math.pow(X, 4);
        double X5 = Math.pow(X, 5);

        double equacoes[] = new double[6];
        equacoes[0] = 1 - (23 * X2 / L2) + (66 * X3 / L3) - (68 * X4 / L4) + (24 * X5 / L5);
        equacoes[1] = X - (6 * X2 / L) + (13 * X3 / L2) - (12 * X4 / L3) + (4 * X5 / L4);
        equacoes[2] = (16 * X2 / L2) - (32 * X3 / L3) + (16 * X4 / L4);
```

```
        equacoes[3] = (-8 * X2 / L) + (32 * X3 / L2) - (40 * X4 / L3) + (16 * X5 / L4);
        equacoes[4] = (7 * X2 / L2) - (34 * X3 / L3) + (52 * X4 / L4) - (24 * X5 / L5);
        equacoes[5] = (-1 * X2 / L) + (5 * X3 / L2) - (8 * X4 / L3) + (4 * X5 / L4);

        int pos = 2 * (elemento.getNoInicial().getNumeroNo() - 1);
        double soma = 0.0;
        for (int i = 0; i < 6; i++) {
            soma += equacoes[i] * deslocamentos[pos + i];
        }

        double deslocada = soma;

        return deslocada;
    }
}

public class Cortante extends VigaAtributo {

    double vetorCortantes[];
    private MatrizNaoInversivelException ex = null;

    public Cortante(Viga viga) {
        super(viga);

        this.vetorCortantes = null;
    }

    public double[] getVetor() throws MatrizNaoInversivelException {
        verificarAtualizacoesViga();

        if (ex != null) {
            throw ex;
        }

        return vetorCortantes;
    }

    @Override
    protected void calcular() {
        try {

            double deslocamentos[] = viga.getDeslocamentos().getVetor();
            int n = viga.getQtdElementos() * 3;

            vetorCortantes = new double[n];

            Elemento el = viga.getElementoInicial();
            while (el != null) {
                int desloc = 3 * (el.getNumeroElemento() - 1);

                vetorCortantes[desloc + 0] = calcCortante(el, 0, deslocamentos);
                vetorCortantes[desloc + 1] = calcCortante(el, el.getL() / 2.0, deslocamentos);
                vetorCortantes[desloc + 2] = calcCortante(el, el.getL(), deslocamentos);

                el = el.getProximo();
            }

        } catch (MatrizNaoInversivelException e) {
            this.vetorCortantes = null;
            this.ex = e;
```

```
        }
    }

    public double getCortanteAt( Elemento elemento, double x ) throws MatrizNaoInversivelException
{
        return calcCortante(elemento, x, viga.getDeslocamentos().getVetor());
    }

    private double calcCortante(Elemento elemento, double x, double deslocamentos[]) {
        double E = elemento.getE();
        double I = elemento.getI();

        double L = elemento.getL();
        double L2 = Math.pow(L, 2);
        double L3 = Math.pow(L, 3);
        double L4 = Math.pow(L, 4);
        double L5 = Math.pow(L, 5);

        double X = x;
        double X2 = Math.pow(X, 2);
        double X3 = Math.pow(X, 3);

        double equacoes[] = new double[6];
        equacoes[0] = 2 * (720 * X2 - 816 * L * X + 198 * L2) / L5;
        equacoes[1] = 2 * (120 * X2 - 144 * L * X + 39 * L2) / L4;
        equacoes[2] = 32 * (12 * X - 6 * L) / L4;
        equacoes[3] = 16 * (12 * L2 - 60 * L * X + 60 * X2) / L4;
        equacoes[4] = 2 * (-720 * X2 + 624 * L * X - 102 * L2) / L5;
        equacoes[5] = 2 * (120 * X2 - 96 * L * X + 15 * L2) / L4;

        int pos = 2 * (elemento.getNoInicial().getNumeroNo() - 1);
        double soma = 0.0;
        for (int i = 0; i < 6; i++) {
            soma += equacoes[i] * deslocamentos[pos + i];
        }

        double cortante = E * I * soma;

        return cortante;
    }
}

public class MomentoFletor extends VigaAtributo {

    double vetorMomentos[];
    private MatrizNaoInversivelException ex = null;

    public MomentoFletor(Viga viga) {
        super(viga);

        this.vetorMomentos = null;

    }

    public double[] getVetor() throws MatrizNaoInversivelException {
        verificarAtualizacoesViga();

        if (ex != null) {
            throw ex;
        }
```

```
        return vetorMomentos;
    }

    @Override
    protected void calcular() {
        try {

            double deslocamentos[] = viga.getDeslocamentos().getVetor();
            int n =  3 * viga.getQtdElementos();

            vetorMomentos = new double[n];

            Elemento el = viga.getElementoInicial();
            while (el != null) {
                //int desloc = 2 * (el.getNumeroElemento() - 1);
                int desloc = 3 * (el.getNumeroElemento() - 1);

                vetorMomentos[desloc + 0] = calcMomentoFletor(el, 0, deslocamentos);
                vetorMomentos[desloc + 1] = calcMomentoFletor(el, el.getL() / 2.0, deslocamentos);
                vetorMomentos[desloc + 2] = calcMomentoFletor(el, el.getL(), deslocamentos);

                el = el.getProximo();
            }

        } catch (MatrizNaoInversivelException e) {
            this.vetorMomentos = null;
            this.ex = e;
        }
    }

    public double getMomentoAt( Elemento elemento, double x ) throws MatrizNaoInversivelException
{
        return calcMomentoFletor(elemento, x, viga.getDeslocamentos().getVetor());
    }

    private double calcMomentoFletor(Elemento elemento, double x, double deslocamentos[]) {
        double E = elemento.getE();
        double I = elemento.getI();

        double L = elemento.getL();
        double L2 = Math.pow(L, 2);
        double L3 = Math.pow(L, 3);
        double L4 = Math.pow(L, 4);
        double L5 = Math.pow(L, 5);

        double X = x;
        double X2 = Math.pow(X, 2);
        double X3 = Math.pow(X, 3);

        double equacoes[] = new double[6];
        equacoes[0] = 2 * (240 * X3 - 408 * L * X2 + 198 * L2 * X - 23 * L3) / L5;
        equacoes[1] = 2 * (40 * X3 - 72 * L * X2 + 39 * L2 * X - 6 * L3) / L4;
        equacoes[2] = 32 * (6 * X2 - 6 * L * X + L2) / L4;
        equacoes[3] = 16 * (2 * X - L) * (10 * X2 - 10 * L * X + L2) / L4;
        equacoes[4] = 2 * (-240 * X3 + 312 * L * X2 - 102 * L2 * X + 7 * L3) / L5;
        equacoes[5] = 2 * (40 * X3 - 48 * L * X2 + 15 * L2 * X - L3) / L4;

        int pos = 2 * (elemento.getNoInicial().getNumeroNo() - 1);
        double soma = 0.0;
```

```
        for (int i = 0; i < 6; i++) {
            soma += equacoes[i] * deslocamentos[pos + i];
        }

        double momento = E * I * soma;

        return momento;
    }

}
public class Reacoes extends VigaAtributo {

    private double[] vetorReacoes;

    private MatrizNaoInversivelException ex = null;

    public Reacoes(Viga viga) {
        super(viga);

        this.vetorReacoes = null;
    }

    public double[] getVetorReacoes() throws MatrizNaoInversivelException {
        verificarAtualizacoesViga();

        if (ex != null) {
            throw ex;
        }

        return vetorReacoes;
    }

    // gera vetor reacoes F
    @Override
    protected void calcular() {
        try {
            this.ex = null; // supõe sem exceção

            double[][] matRigidezGlobal = viga.getMatrizRigidez().getMatriz();
            double[] vetorDeslocamentos = viga.getDeslocamentos().getVetor();

            int n = vetorDeslocamentos.length;

            double F[] = new double[n];

            //multiplicação de matrizes
            for (int i = 0; i < n; i++) {
                double soma = 0.0;
                for (int j = 0; j < n; j++) {
                    soma += matRigidezGlobal[i][j] * vetorDeslocamentos[j];
                }
                F[i] = soma;
            }


            this.vetorReacoes = F;

        } catch (MatrizNaoInversivelException ex) {
            this.vetorReacoes = null;
            this.ex = ex;
```

```
        }
    }

    public double getReacaoForca(No no) throws MatrizNaoInversivelException {
        if (no.isDeslocamentoY()) {
            int pos = 2 * (no.getNumeroNo() - 1);

            return getVetorReacoes()[pos] - viga.getCargas().getVetor()[pos];
        } else {
            return 0.0;
        }
    }

    public double getReacaoMomento(No no) throws MatrizNaoInversivelException {
        if (no.isGiro()) {
            int pos = 1 + (2 * (no.getNumeroNo() - 1));

            return getVetorReacoes()[pos] - viga.getCargas().getVetor()[pos];
        } else {
            return 0.0;
        }
    }
}
public class Viga {
    /* informações da viga */

    private double L;
    private final MatrizRigidezGlobal matrizRigidez;
    private final CargasGlobal cargas;
    private final SolucaoContorno solucaoContorno;
    private final Reacoes reacoes;
    private final Deslocamentos deslocamentos;
    private final MomentoFletor momentoFletor;
    private final Cortante cortante;
    private final Deslocada deslocada;

    /* navegação */
    private Elemento elementoInicial;
    private Elemento elementoFinal;
    private int qtdElementos;

    /* controle alteracoes */
    private int versaoConteudo;


    /* */
    public Viga() {
        this.elementoInicial = null;
        this.elementoFinal = null;
        this.L = 0;
        this.qtdElementos = 0;

        this.matrizRigidez = new MatrizRigidezGlobal(this);
        this.cargas = new CargasGlobal(this);
        this.solucaoContorno = new SolucaoContorno(this);
        this.reacoes = new Reacoes(this);
        this.deslocamentos = new Deslocamentos(this);
        this.momentoFletor = new MomentoFletor(this);
        this.cortante = new Cortante(this);
        this.deslocada = new Deslocada(this);
```

```
    this.versaoConteudo = 1;
}

// retorna a versao do conteudo da viga
public int getVersaoConteudo() {
    return this.versaoConteudo;
}

public boolean isVigaAlterada(int versaoConteudo) {
    return this.versaoConteudo != versaoConteudo;
}

public void vigaAlterada() {
    versaoConteudo++;
}

public void incorporarElementoAfter(Elemento novoEl, Elemento refEl) {
    // verifica se elemento realmente novo, e
    // e se o elemento de referência já faz mesmo parte da lista
    if (novoEl.getNumeroElemento() <= 0 && refEl.getNumeroElemento() > 0) {
        if (elementoFinal == refEl) {
            elementoFinal = novoEl;
        }

        novoEl.proximo = refEl.proximo;
        novoEl.anterior = refEl;
        if (refEl.proximo != null) {
            refEl.proximo.anterior = novoEl;
        }
        refEl.proximo = novoEl;

        // propaga modificações na viga
        incorporarElemento(refEl);
    }

}

/* adiciona um novo elemento na viga (FILA) */
public void incorporarElemento(Elemento el) {

    if (el.getNumeroElemento() > 0) {
        // elemento ja existe -> atualizar viga

        // retorocede caracteristicas da viga a estado antes do elemento
        this.L = el.getX();
        this.qtdElementos = el.getNumeroElemento() - 1;

        // Atualizar elementos atinidos
        Elemento pEl = el;
        while (pEl != null) {
            // atualiza características do elemento como parte da viga
            pEl.setX(this.L);
            pEl.setNumeroElemento(qtdElementos + 1);

            // atualiza características da viga
            this.L += pEl.getL();
            this.qtdElementos++;

            pEl = pEl.proximo;
```

```
            }
        } else {
            // elemento novo
            if (this.elementoFinal == null) {
                this.elementoInicial = el;
                this.elementoFinal = el;
            } else {
                this.elementoFinal.proximo = el;
                el.anterior = this.elementoFinal;
                this.elementoFinal = el;

                if (this.elementoInicial == null) {
                    this.elementoInicial = el;
                }
            }

            // atualiza características do elemento como parte da viga
            el.setX(this.L);
            el.setNumeroElemento(this.qtdElementos + 1);

            // atualiza características da viga com entrada do novo elemento
            this.L += el.getL();
            this.qtdElementos++;
        }

        // garante integridade de nós sobrepostos
        // que devem possuir Giro e Deslocamento coincidentes
        if (el.anterior != null) {
            el.anterior.getNoFinal().copiar(el.getNoInicial());
            //el.anterior.setNoFinal(el.getNoInicial());
        }
        if (el.proximo != null) {
            //el.proximo.setNoInicial(el.getNoFinal());
            el.proximo.getNoInicial().copiar(el.getNoFinal());
        }

        vigaAlterada();
    }

    public void removerElemento(Elemento el) {
        if (el.getNumeroElemento() > 0) {

            // retorocede caracteristicas da viga a estado antes do elemento
            this.L = el.getX();
            this.qtdElementos = el.getNumeroElemento() - 1;

            // Atualizar elementos atingidos
            Elemento pEl = el.proximo;
            while (pEl != null) {
                // atualiza características do elemento como parte da viga
                pEl.setX(this.L);
                pEl.setNumeroElemento(this.qtdElementos + 1);

                // atualiza características da viga
                this.L += pEl.getL();
                this.qtdElementos++;

                pEl = pEl.proximo;
            }
```

```
        // refaz ligações entre elementos
        if (el == elementoInicial) {
            elementoInicial = el.proximo;
        }
        if (el == elementoFinal) {
            elementoFinal = el.anterior;
        }
        if (el.anterior != null) {
            el.anterior.proximo = el.proximo;
        }
        if (el.proximo != null) {
            el.proximo.anterior = el.anterior;
        }

        // caso a saida conecte dois elementos...
        if (el.anterior != null && el.proximo != null) {
            // garante integridade de nós sobrepostos
            // que devem possuir Giro e Deslocamento coincidentes
            el.anterior.getNoFinal().copiar(el.proximo.getNoInicial());
            // - - -
        }

        // elemento ja existe -> atualizar viga
        el.setNumeroElemento(0); // marca como fora da viga

        vigaAlterada();
    }
}

public double getL() {
    return L;
}

public Elemento getElementoInicial() {
    return elementoInicial;
}

public Elemento getElementoFinal() {
    return elementoFinal;
}

public Elemento getElementoAt(double x) {
    // força x nos limites da viga
    if (x < 0.0) {
        x = 0.00;
    } else if (x > getL()) {
        x = getL();
    }

    Elemento el = elementoInicial;
    double comp = 0;
    while (el != null) {
        comp += el.getL();

        if (x <= comp) {
            return el;
        }

        el = el.getProximo();
    }
```

```
    return null;
}

public int getQtdElementos() {
    return qtdElementos;
}

public int getQtdNos() {
    return (qtdElementos > 0) ? ((qtdElementos * 2) + 1) : 0;
}

public MatrizRigidezGlobal getMatrizRigidez() {
    return matrizRigidez;
}

public CargasGlobal getCargas() {
    return cargas;
}

public SolucaoContorno getSolucaoContorno() {
    return solucaoContorno;
}

public Reacoes getReacoes() {
    return reacoes;
}

public Deslocamentos getDeslocamentos() {
    return deslocamentos;
}

public MomentoFletor getMomentoFletor() {
    return momentoFletor;
}

public Cortante getCortante() {
    return cortante;
}

public Deslocada getDeslocada() {
    return deslocada;
}public abstract class VigaAtributo {
private int lastVigaUpadate;
protected final Viga viga;

public VigaAtributo(Viga viga) {
    this.viga = viga;
    lastVigaUpadate = viga.getVersaoConteudo() -1 ;
}

// recalcula sempre que viga sofreu alteracoes
// desde últmo cálculo
protected void verificarAtualizacoesViga() {
    if (viga.isVigaAlterada(lastVigaUpadate)) {
        this.calcular();
        this.lastVigaUpadate = viga.getVersaoConteudo();
    }
}
```

```
    // (re)calcula com valores da viga atual
    protected abstract void calcular();
}

}
```