UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA CIVIL

Desenvolvimento de software para engenharia de estruturas: calculadora de flechas imediatas em vigas biapoiadas de concreto armado

PEDRO FERRARI GALAN DÉO

São Carlos
2022

PEDRO FERRARI GALAN DÉO

DESENVOLVIMENTO DE SOFTWARE PARA ENGENHARIA DE ESTRUTURAS: CALCULADORA DE FLECHAS IMEDIATAS EM VIGAS BIAPOIADAS DE CONCRETO ARMADO

Trabalho de conclusão de curso apresentado como requisito parcial para a obtenção do título de Bacharel em Engenharia Civil pela Universidade Federal de São Carlos.

Orientador: Guilherme Aris Parsekian

São Carlos

2022

Déo, Pedro Ferrari Galan.
Nº Cutter Desenvolvimento de software para engenharia de estruturas: calculadora de flechas imediatas em vigas biapoiadas de concreto armado / Pedro Ferrari Galan Déo. — 2022.
84 f.

Trabalho de Conclusão de Curso (Graduação) – Universidade Federal de São Carlos, campus São Carlos, 2022

1. Flecha imediata. 2. Vigas. 3. Concreto armado. I. Título.

CDD [número da CDD].

DESENVOLVIMENTO DE SOFTWARE PARA ENGENHARIA DE ESTRUTURAS: CALCULADORA DE FLECHAS IMEDIATAS EM VIGAS BIAPOIADAS DE CONCRETO ARMADO.

Pedro Ferrari Galan Déo

Trabalho de conclusão de curso apresentado como requisito parcial para a obtenção do título de Bacharel em Engenharia Civil pela Universidade Federal de São Carlos.

Aprovado em: _____/_____/_____.

BANCA EXAMINADORA

_____

Orientador

Guilherme Aris Parsekian

UFSCar

_____

Membro da banca (1)

Fernando Menezes de Almeida Filho

UFSCar

_____

Membro da banca (2)

Ricardo Laguardia Justen de Almeida

UFSCar

Dedico este trabalho a todo o curso de Engenharia Civil da Universidade Federal de São Carlos, professores e colegas, que me ajudaram ao longo desta etapa. E aos meus pais, pois para mim eles sempre serão meus maiores professores.

RESUMO

As vigas de concreto armado são amplamente estudadas por alunos e profissionais da engenharia civil. Por ser o concreto armado um dos sistemas estruturais mais utilizados nas obras executadas, torna-se essencial que os cálculos utilizados em seu dimensionamento sejam precisos, visando aproveitar ao máximo os materiais utilizados em sua composição. Quando submetidos à flexão, fissuras se formam no elemento estrutural, o que causa uma redução na rigidez da viga. É fundamental a consideração desta não linearidade física do concreto para obter resultados próximos ao caso real quando se calcula o deslocamento de vigas. Métodos para o cálculo deste deslocamento são estudados até hoje, porém dentre eles podemos destacar dois métodos simplificados, o de Branson (1968), adotado pela NBR 6118 (ABNT, 2014), e o método de Bischoff (2005), utilizado pela ACI 318-19 (American Concret Institute, 2019). Outro método amplamente estudado utiliza elementos finitos para simular uma viga em concreto armado. Este trabalho buscou criar uma ferramenta computacional, utilizando a linguagem Python, capaz de calcular o deslocamento em vigas utilizando todos estes métodos simultaneamente, permitindo a comparação entre os resultados obtidos através de gráficos e tabelas.

Palavras-chave: flecha imediata, vigas, concreto armado, não-linearidade física

ABSTRACT

Reinforced concrete beams are widely studied by the civil engineering students and professionals. Since reinforced concrete is one of the most used structural systems, it becomes essential that the calculations used in its design are precise, aiming to use maximum capacity of the composing materials. When under bending stress, cracks begin to form in the structural element, causing a reduction in beam stiffness. It is fundamental to consider this physical nonlinearity of concrete to obtain results close to the real case when computing deflection in beams. Calculation methods to obtain this deflection are studied until today, although among them two direct methods stand out, Branson's (1985), adopted by NBR 6118 (ABNT, 2014), and Bischoff's (2005), used by the ACI 318-19 (American Concrete Institute, 2019). Another vastly studied method utilizes finite elements to simulate a reinforced concrete beam. This paper pursue consists in developing a software, using the Python language, capable of calculating beam deflections utilyzing all these methods simultaneously, allowing the comparsion between the obtained results through graphics and charts.

Keywords: instantaneous deflection, beams, reinforced concrete, physical nonlinearity

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

Até hoje, a maior parte das estruturas executadas na construção civil são feitas em concreto armado. Ao combinar o concreto e o aço, cria-se um composto que é resistente à compressão e tração, além de ser um material que pode ser moldado praticamente em qualquer forma. O ato de projetar uma estrutura em concreto armado é um processo complexo que envolve uma vasta quantidade de variáveis. Cabe ao engenheiro projetista estabelecer parâmetros e desenvolver uma concepção estrutural que seja compatível com os projetos, muitas vezes já existentes, de arquitetura e instalações prediais. Como as cargas atuantes na estrutura só são obtidas após o anteprojeto estrutural, é necessário então obter os esforços e deslocamentos que atuam na estrutura através de um modelo de cálculo.

Ao realizar a verificação do anteprojeto, são obtidos os valores de tensão e deformação para cada um dos elementos que compõem a estrutura, como lajes, vigas e pilares. Estes valores mostram ao projetista se a estrutura suporta o carregamento existente, se há necessidade de aumentar a capacidade resistiva da estrutura, ou se é possível diminuir esta capacidade para obter uma economia de material.

A norma brasileira NBR 6118:2014 (ABNT, 2014) estipula limites a serem seguidos durante o cálculo e a verificação de estruturas em concreto armado, como taxas de armadura mínimas e máximas, deformações e resistência mecânica máximas para elementos em situações de serviço e de estado limite último. Com a atual utilização de elementos cada vez mais esbeltos na engenharia de estruturas, torna-se cada vez mais relevante o cálculo preciso da deformação destes elementos, para garantir o cumprimento destes limites e aproveitar ao máximo a capacidade mecânica dos materiais envolvidos.

A utilização de métodos computacionais para a realização destes cálculos é cada vez mais preciosa para o mercado. No Brasil, muitos softwares são utilizados para o cálculo e verificação de estruturas, como TQS, Eberick, Cypecad, SAP2000, FTOOL. O conceito utilizado pelos programas de cálculo é o método dos elementos finitos, onde uma estrutura é decomposta em pequenos elementos. Para tal, são utilizados modelos

matemáticos para representar a estrutura, como a análise matricial utilizando analogias de barras, grelhas ou cascas. A escolha do modelo computacional que melhor representa a estrutura em estudo é fundamental para garantir a precisão dos cálculos e a otimização do tempo de processamento.

A complexidade de modelar uma estrutura aumenta quando é considerada a não linearidade física dos materiais utilizados. O concreto e o aço quando submetidos a tensões apresentam comportamento elástico até um certo nível de carregamento, porém esta linearidade é perdida quando as tensões superam os limites de elasticidade do material. Para criar modelos computacionais compatíveis com o caso real é necessário considerar esta perda de elasticidade. Um método utilizado para tal é o do carregamento incremental, onde o elemento estudado é submetido a um carregamento crescente e a cada incremento é realizada a integralização das tensões existentes, caso o concreto apresente fissuras.

Além de métodos iterativos, pode-se considerar a não-linearidade física do concreto utilizando formulações simplificadas ou diagramas de momento-curvatura das seções estudadas. Caso esses diagramas sejam previamente calculados, o esforço computacional pode ser reduzido significativamente. Ainda assim, neste trabalho optou-se por utilizar os métodos incrementais, visto que esta diferença de performance entre os dois métodos se mostra pouco significativa, devido aos avanços da tecnologia dos computadores atuais. Há alguns anos, rotinas de cálculo incremental eram mais difíceis de serem implementadas computacionalmente, devido às limitações de memória e capacidade de processamento dos computadores. Entretanto, mesmo com a grande evolução dos processadores de hoje, o método dos elementos finitos se torna exponencialmente mais lento ao passo que se aumentam o número de elementos utilizados na modelagem.

Métodos como o incremental descrito acima, que simulam o caso real do carregamento da viga, são chamados de métodos exatos, na medida em que permitem verificar o comportamento do elemento para diferentes fases de carregamento e condições de contorno. Além desses, existem métodos simplificados como propostos por Branson (1968) e Bischoff (2005), que fornecem uma maneira rápida de realizar o cálculo do deslocamento em vigas de forma analítica, obtendo resultados próximos ao caso real.

Devido a facilidade de realização destes cálculos e sua boa precisão, esses métodos foram adotados pela norma brasileira NBR 6118 (ABNT, 2014) e pela norma norte americana ACI 318-R19 (*American Concrete Institute*, 2019). A ideia é estimar uma inércia efetiva em vigas fissuradas de concreto armado no projetar de estruturas usuais.

Este trabalho buscou desenvolver uma aplicação, implementada na linguagem Python, capaz de simular carregamento incremental em vigas de concreto armado, para previsão de flechas imediatas. Para este cálculo, foram utilizados os métodos previstos nas normas anteriormente citadas, e o método dos elementos finitos com correção da inércia efetiva após cada etapa de carga. Os resultados de cada etapa de carregamento foram organizados em gráficos comparativos, possibilitando a visualização dos diferentes resultados obtidos para cada método

Utilizando o software desenvolvido, foram resolvidos exemplos para apresentar as possíveis utilizações desta ferramenta computacional, apresentados no capítulo 4. Nota-se que o script desenvolvido possui aplicação prática para a engenharia civil, sendo utilizável na verificação dos estados limite de serviço de vigas de concreto armado quanto a deformação de acordo com a NBR 6118 (ABNT, 2014).

1.1. Objetivos

O principal objetivo do presente trabalho é desenvolver um aplicativo capaz de realizar a estimativa da flecha imediata em vigas de concreto armado utilizando três métodos diferentes de cálculo, considerando a não linearidade física do concreto após a sua fissuração. Para isto foi desenvolvido um software em linguagem Python, intitulado “ImediataVigas”, capaz de calcular esforços e deslocamentos em cada ponto da viga, utilizando o método dos elementos finitos, e métodos utilizando a inércia efetiva de Branson (1968) e a de Bischoff (2005), para serem salvos em arquivos de tabelas e apresentados através de gráficos.

Os objetivos específicos são:

• Revisão de literatura sobre os materiais envolvidos, os métodos de análise citados e sobre a linguagem de programação escolhida;

• Desenvolver um aplicativo em linguagem Python com interface gráfica capaz de calcular o deslocamento em vigas de concreto armado e apresentar os resultados obtidos, utilizando os valores da inércia efetiva de Branson (1968) e Bischoff (2005), além do método dos elementos finitos utilizando carregamento incremental, como proposto por Carvalho (1994);

• Apresentar a capacidade de utilização da ferramenta computacional desenvolvida com a resolução de exemplos.

1.2. Justificativa

O estudo da deflexão em vigas de concreto armado é assunto de inúmeras pesquisas no campo da Engenharia Civil, bem como a utilização de métodos computacionais para resolver estruturas reticuladas. É importante o desenvolvimento de novas ferramentas computacionais que possibilitem o estudo comparativo entre os métodos existentes, para auxiliar em tomadas de decisão no ato de projetar e no desenvolver de pesquisas futuras.

A importância em obter os resultados do deslocamento de vigas, reunir estes dados e compará-los com os equacionamentos sugeridos pelas normas nacionais e internacionais, vem da procura por métodos de cálculo mais precisos, que visam otimizar o uso dos materiais envolvidos para aproveitar ao máximo suas capacidades mecânicas.

Por último, o valor deste trabalho reside inclusive em sua capacidade de reutilização, seja da aplicação apresentada ou de seu código fonte, auxiliando no desenvolvimento de um futuro software mais robusto.

2. REVISÃO BIBLIOGRÁFICA

O concreto utilizado na construção civil é composto principalmente por água, cimento e agregados. Para o cálculo da resistência mecânica dos elementos compostos por este material, é necessário informações sobre o tipo de cimento e o agregado utilizado na sua composição. Por ser um material que apresenta ruptura frágil, além de baixa resistência a tração quando comparada ao valor resistido na compressão, o concreto quase sempre está associado a outro material, sendo o aço o mais utilizado nesta composição. Ao executar um elemento estrutural em concreto armado, com o correto dimensionamento das armaduras, os esforços de compressão na estrutura são resistidos pelo concreto e os de tração pelo aço, formando um sistema estrutural que possui como principais características o trabalho em conjunto do concreto e do aço decorrente da aderência entre os materiais, e a possível formação de fissuras em trechos de seus elementos constituintes. (CARVALHO E FIGUEIREDO FILHO, 2014).

Sendo o foco deste trabalho os elementos de viga compostos de concreto armado por barras de aço, a seguir serão apresentadas características, métodos e conceitos utilizados no dimensionamento e na verificação destes elementos estruturais.

2.1. Concreto

No Brasil, ao projetar uma estrutura em concreto armado, o engenheiro precisa seguir os procedimentos apresentados na norma brasileira NBR 6118 (ABNT, 2014), que fornece equacionamento para se obter, em função da resistência à compressão do concreto (f_{ck}), as demais resistências do concreto, no caso da falta de ensaios que comprovem os valores reais destas resistências. A seguir são apresentadas as propriedades do concreto pertinentes ao desenvolvimento deste trabalho.

2.1.1. Massa específica

A NBR 6118 (ABNT, 2014) indica que o concreto, quando seco em estufa, apresenta um valor de massa específica (ρ_c) variando entre 2000 kg/m³ e 2800 kg/m³, e

afirma que na ausência de ensaios que comprovem este valor, pode-se adotar os valores de 2400 kg/m³ para o concreto simples e 2500 kg/m³ para o concreto armado.

2.1.2. Coeficiente de Dilatação Térmica

Na análise estrutural, pode-se utilizar o valor de 10^{-5}/ºC para o coeficiente de dilatação térmica do concreto (α), conforme sugerido pela NBR 6118 (ABNT, 2014).

2.1.3. Resistência à Compressão

A resistência à compressão do concreto (f_c) é obtida através de ensaios conduzidos de acordo com a NBR 6118 (ABNT, 2014). Já a sua resistência característica (f_{ck}), é o valor de resistência que define um intervalo de confiança de 95% para a média dos valores de resistência obtidos através de ensaios (CARVALHO E FIGUEIREDO FILHO, 2014). Por fim, a NBR 8953 (ABNT, 2015) classificou os tipos de concreto estrutural em dois grupos, conforme a sua resistência característica à compressão (TABELA 1).

Tabela 1 – Classes de resistência de concretos estruturais do grupo I

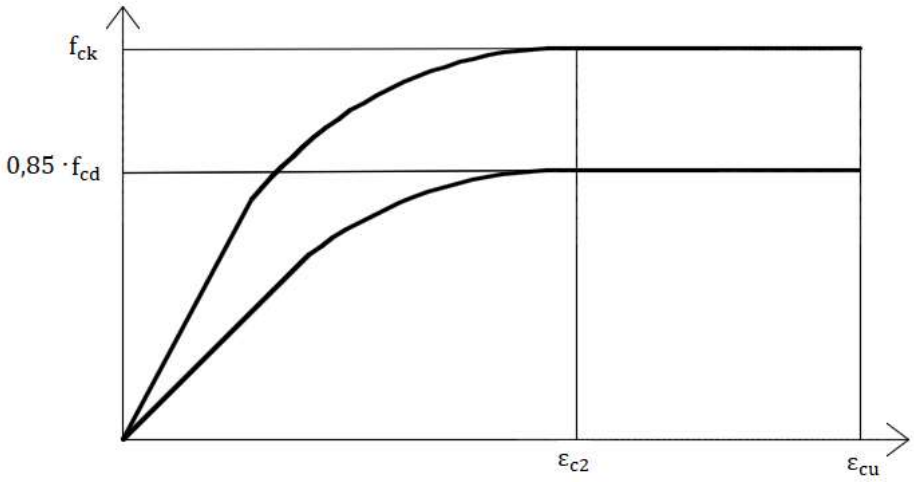
Classe de resistência	Resistência Característica à compressão (MPa)
C20	20
C25	25
C30	30
C35	25
C40	40
C45	45
C50	50

Fonte: Adaptado de NBR 8953 (ABNT, 2015)

Ainda, ao dimensionar uma estrutura para o estado limite último da seção transversal de elementos em concreto armado, é recomenda a utilização do diagrama

idealizado tensão-deformação do concreto submetido à compressão, fornecida pela NBR 6118 (ABNT, 2014), no item 8.2.10.1 (FIGURA 1).

Figura 1 – Diagrama tensão-deformação idealizado do concreto comprimido



Fonte: Adaptado de NBR 6118 (ABNT, 2014)

Os valores a serem utilizados para ε_{c2} e ε_{cu}, de acordo com a NBR 6118 (ABNT, 2014) são:

Para concretos de classes até C50:

$$\varepsilon_{c2} = 2‰ \quad (1)$$

$$\varepsilon_{cu} = 3{,}5‰ \quad (2)$$

Sendo:

ε_{c2} a deformação específica de encurtamento do concreto no início do patamar plástico, e

ε_{cu} a deformação específica de encurtamento do concreto na ruptura.

2.1.4. Resistência à Tração

Em alguns elementos estruturais, o material é submetido majoritariamente a esforços de tração, como a tração na flexão em vigas ou a tração pura em tirantes. Como a fissuração de uma peça ocorre devido a tensões de tração superiores à capacidade

mecânica do concreto, a resistência a tração deste material torna-se relevante à durabilidade destes elementos estruturais (OLUKUN, 1991).

Na falta de ensaios para obtenção da resistência a tração do concreto (f_{ct}), o seu valor médio ou característico pode ser calculado de acordo com o equacionamento abaixo, apresentado no item 8.2.5 da NBR 6118 (ABNT, 2014).

$$f_{ctk,inf} = 0{,}7 \cdot f_{ct,m} \quad (3)$$

$$f_{ctk,sup} = 1{,}3 \cdot f_{ct,m} \quad (4)$$

Onde:

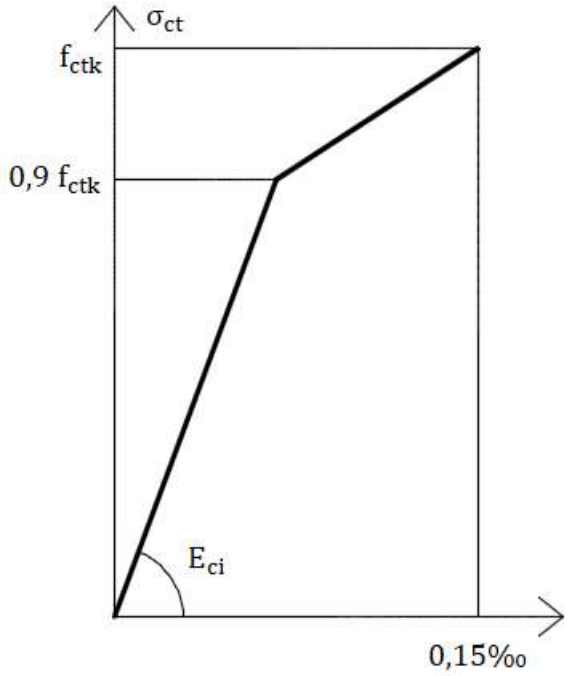
$$f_{ct,m} = 0{,}3 \cdot f_{ck}^{\,2/3} \text{ concretos de classes até C50;} \quad (5)$$

Sendo:

$f_{ct,m}$ e f_{ck} são dados em mega pascal (MPa).

Antes da formação de fissuras de tração no concreto, a NBR 6118 (ABNT, 2014) estabelece a adoção do diagrama tensão-deformação bi linear de tração, em seu item 8.2.10.2 (FIGURA 2).

Figura 2 – Diagrama tensão-deformação bi linear de tração do concreto



Fonte: Adaptado de ABNT NBR 6118 (2014)

2.1.5. Módulo de Elasticidade

O valor do módulo de elasticidade inicial (E_{ci}) do concreto pode ser obtido através do equacionamento presente no item 8.2.8 da NBR 6118 (ABNT, 2014), como apresentado abaixo.

$$E_{ci} = \alpha_e \cdot 5600 \cdot \sqrt{f_{ck}} \quad \text{para concretos de classes até C50} \quad (6)$$

Onde:

$$\alpha_e = \begin{cases} 1{,}2 \text{ para basalto e diabásio;} \\ 1{,}0 \text{ para granito e gnaisse;} \\ 0{,}9 \text{ para calcário;} \\ 0{,}7 \text{ para arenito;} \end{cases}$$

Sendo:

E_{ci} e f_{ck} são dados em megapascal (MPa);

Ainda, para estimar o módulo de deformação secante (E_{cs}), pode-se utilizar equação 7, conforme o mesmo item supracitado.

$$E_{cs} = \alpha_i \cdot E_{ci} \quad (7)$$

Sendo:

$$\alpha_i = 0{,}8 + 0{,}2 \cdot \frac{f_{ck}}{80} \leq 1{,}0 \quad (8)$$

2.1.6. Coeficiente de Poisson e módulo de elasticidade transversal

Em seu item 8.2.9, a NBR 6118 (ABNT, 2014) pontua que, para tensões de compressão menores que 50% da resistência à compressão do concreto (f_c) e tensões de tração menores que a resistência a tração do concreto (f_{ct}), o coeficiente de Poisson (ν) pode ser utilizado com o valor de 0,2, e o módulo de elasticidade transversal do concreto (G_c) pode ser obtido pela equação 9.

$$G_c = \frac{E_{cs}}{2{,}4} \quad (9)$$

2.2. Aço

Aço é um material amplamente utilizado na construção civil por apresentar alta resistência a tração, compressão, flexão e torção. Embora existam outras aplicações para este material, o presente estudo foi limitado ao estudo de barras de aço utilizadas para armadura de concreto, conforme a NBR 7480 (ABNT, 2007). Esta mesma norma define quais tipos de barra devem ser usadas como armaduras para concreto armado, e a resistência característica de escoamento (f_{yk}) de cada uma das categorias de barras (TABELA 2).

Tabela 2 – Propriedades mecânicas de barras e fios de aço destinados a armaduras para concreto armado

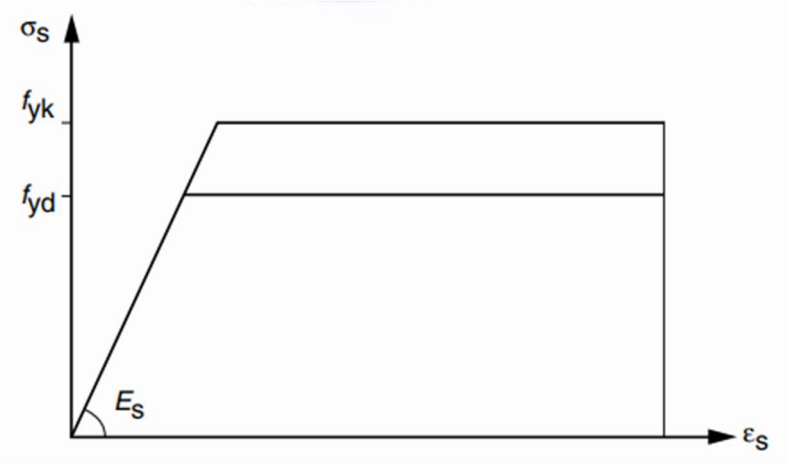
Categoria	**Resistência característica de escoamento** (MPa)
CA-25	250
CA-50	500
CA-60	600

Fonte: Adaptado de NBR 7480 (ABNT, 2007)

2.2.1. Resistência ao escoamento

No dimensionamento de estruturas, para os estados limites de serviço e último, o valor da tensão de escoamento do aço (f_y) pode ser obtido utilizando o diagrama tensão-deformação simplificado, para aços com ou sem patamar de escoamento (FIGURA 3).

Figura 3 – Diagrama tensão-deformação para aços de armadura passiva



Fonte: Adaptado de NBR 6118 (ABNT, 2014)

2.2.2. Módulo de Elasticidade

Na falta de informações do fabricante ou de ensaios, a NBR 6118 (ABNT, 2014) admite o módulo de elasticidade do aço de armadura passiva (E_s) igual a 210 GPa.

2.3. Cálculo Estrutural

Carvalho e Figueiredo Filho (2014) afirmaram que calcular uma estrutura significa garantir que ela suporte a suas solicitações com segurança, e atenda a limites impostos por normas para as deformações de seus elementos. As deformações em excesso podem causar o colapso total da estrutura, ou impossibilitar a sua utilização. Para a

padronização do dimensionamento, a NBR 6118 (ABNT, 2014) impõe a verificação de estados-limites últimos e de serviço, garantindo que as solicitações de cálculo sejam menores do que a capacidade resistente da estrutura. Nesse método, as resistências são minoradas e as ações são majoradas, com coeficientes de ponderação específicos.

2.3.1. Valores de cálculo das resistências

Segundo a NBR 6118 (ABNT, 2014), obtém-se o valor de cálculo das capacidades resistivas de materiais (f_d) através da equação 12.

$$f_d = \frac{f_k}{\gamma_m} \quad (10)$$

Onde:

f_k é a resistência característica do material,

γ_m é o coeficiente de ponderação relativo ao material.

2.3.2. Valores de cálculo das ações

De maneira similar ao item anterior, os valores de cálculo são obtidos através da multiplicação de seus valores característicos por coeficientes de ponderação, a fim de majorar seu valor.

2.3.3. Estados Limites

Ao dimensionar estruturas de concreto armado, são levados em conta os estados limites último e os estados limites de serviço. A seguir são listados os estados a serem sempre analisados no projetar de uma estrutura.

2.3.3.1. Estados Limites Últimos (ELU)

Os casos (a) a (g), apresentados em sequência, precisam ser verificados para garantir a segurança contra ruína das estruturas de concreto armado. São estes estados limites últimos:

a) Perda do equilíbrio estrutural;
b) Esgotamento da capacidade resistiva da estrutura, em partes ou em sua totalidade, devido a solicitações normais ou tangenciais;
c) Esgotamento da capacidade resistiva da estrutura, em partes ou em sua totalidade, porém considerando os efeitos de segunda ordem;
d) Provocado por solicitações dinâmicas;
e) Colapso progressivo;
f) Esgotamento da capacidade resistente da estrutura, em partes ou em sua totalidade, considerando situação de exposição a incêndio.
g) Esgotamento da capacidade resistente da estrutura, em partes ou em sua totalidade, considerando atuação de sismos.

2.3.3.2. Estados Limites de Serviço (ELS)

a) Estado Limite de Formação de Fissuras (ELS-F): o estado em que surgem as primeiras fissuras no concreto;
b) Estado Limite de Abertura de Fissuras (ELS-W): neste estado as fissuras possuem aberturas iguais aos valores máximos especificados pela norma (ABNT, 2014).
c) Estado Limite de Deformação Excessiva (ELS-DEF): a norma oferece limites para a deformação dos elementos estruturais de concreto armado. Neste estado, ocorrem deformações iguais a estes valores.
d) Estado Limite de Vibrações Excessivas (ELS-VE): estado onde as vibrações atingem seus valores limites para a utilização da construção.

2.3.4. Ações

As ações que atuam nas estruturas são classificadas de acordo com período na qual elas atuam na estrutura, sendo estas:

a) Ações permanentes: são ações que estão sempre atuando na estrutura, durante toda sua vida útil. Estas podem ser divididas em diretas, como o peso

próprio da estrutura, e indiretas, como deslocamentos impostos aos apoios, imperfeições geométricas, retração e fluência do concreto;

b) Ações variáveis: são as cargas acidentais, que atuam apenas durante uma parte da vida da estrutura. Podem ser provenientes do uso da construção, das forças do vento ou da água, e por variações de temperatura da estrutura.
c) Ações excepcionais: São cargas que ocorrem de forma inesperada durante um tempo muito curto, como incêndios, choques com a estrutura, explosões de gás ou de outras origens.

2.3.5. Combinações das ações

O processo de dimensionar uma estrutura visa garantir que ela suporte todas as ações solicitantes que atuarão durante o seu processo de execução e sua vida útil. Neste contexto, define-se um carregamento pela combinação das ações que atuam simultaneamente numa estrutura, em um período estabelecido. O intuito de analisar diferentes combinações de ações é determinar o caso mais desfavorável para a estrutura estudada (CARVALHO E FIGUEIREDO FILHO, 2014).

As respectivas combinações de ações para verificar cada estado limite último e de serviço são apresentadas pela NBR 6118 (ABNT, 2014). A seguir, são apresentadas as combinações pertinentes ao desenvolvimento deste trabalho.

2.3.5.1. Combinações últimas normais

Para cada combinação última normal, considera-se predominante uma das ações variáveis que atua na estrutura, considerando que esta atua com seu valor característico. Caso haja outras cargas acidentais, estas são consideradas secundárias e suas intensidades são minoradas por um coeficiente de combinação. Esta ponderação é feita através da equação 13.

$$F_d = \gamma_g \cdot F_{gk} + \gamma_q \cdot \left[F_{q1k} + \sum\left(\psi_{0j}\, F_{qjk}\right)\right] \qquad (11)$$

Onde:

F_d o valor de cálculo das ações para a combinação última;

F_{gk} a soma das ações permanentes diretas;

F_{qk} as ações variáveis diretas, sendo F_{q1k} a principal;

γ_g, γ_q os coeficientes de ponderação de ações permanentes e acidentais, respectivamente;

ψ_{0j} o fator de redução de combinação para o estado limite último.

2.3.5.2. Combinações de serviço usuais

A norma brasileira (ABNT, 2014) oferece o equacionamento para três tipos de combinação de ações para situações de serviço em estruturas usuais de concreto armado (TABELA 3).

Tabela 3 – Combinações de Serviço

	Descrição	Equacionamento
Combinação quase permanente de serviço (CQP)	Todas as ações variáveis são consideradas com seus valores quase permanentes $\psi_2 F_{qk}$.	$F_{d,ser} = \Sigma F_{gik} + \Sigma\psi_{2j} F_{qjk}$ (12)
Combinação frequente de serviço (CF)	A ação variável principal é considerada com seu valor frequente ($\psi_1 F_{q1k}$), e as demais são consideradas com seus valores quase permanentes ($\psi_2 F_{qk}$).	$F_{d,ser} = \Sigma F_{gik} + \psi_1 F_{q1k} + \Sigma\psi_{2j} F_{qjk}$ (13)
Combinação rara de serviço (CR)	A ação principal é tomada com seu valor característico (F_{q1k}), e todas as outras com seus valores frequentes ($\psi_1 F_{q1k}$).	$F_{d,ser} = \Sigma F_{gik} + F_{q1k} + \Sigma\psi_{1j} F_{qjk}$ (14)

Fonte: Adaptado de NBR 6118 (ABNT, 2014)

2.4. Vigas em Concreto Armado

Segundo a NBR 6118 (ABNT, 2014), as vigas são elementos estruturais que estão submetidos principalmente a esforços solicitantes de flexão. Além das propriedades dos materiais envolvidos na composição da estrutura, a seguir são apresentados métodos, considerações e equações necessárias para estudo e dimensionamento destes elementos.

2.4.1. Tipos de flexão

A flexão, oriunda do momento fletor solicitante de um elemento, causa o surgimento de tensões normais (tração e compressão) nas seções que o compõem. Para estudar o funcionamento destes elementos, é necessário identificar o tipo de flexão a qual a estrutura está submetida. Os conceitos para classificar cada um destes tipos de flexão, de acordo com Carvalho e Figueiredo Filho (2014) são apresentados a seguir.

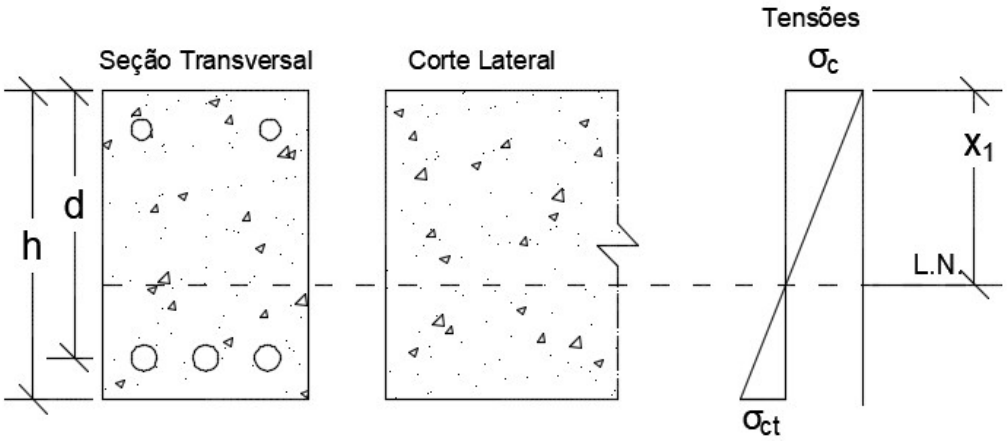
a) **Flexão normal** é quando o plano de atuação da resultante dos carregamentos aplicado na estrutura contém um dos eixos principais de inércia da seção.
b) **Flexão oblíqua** é quando o plano de atuação da resultante dos carregamentos não contém um dos eixos principais de inércia da seção, por possuir armaduras assimétricas, por exemplo.
c) **Flexão simples** é quando não há esforço normal atuando na seção.
d) **Flexão composta** é a situação em que há esforços normais atuando na seção.
e) **Flexão pura** trata-se de um caso particular da flexão, em que a força cortante na seção é nula e o momento fletor é constante.

Com exceção das vigas com armaduras ativas, inicialmente pode-se considerar que estes elementos estão submetidos a flexão normal, simples e pura, o que permite o dimensionamento da armadura longitudinal, aquela que resiste aos esforços de tração provenientes da flexão, pelo menos para a seção mais solicitada do elemento estrutural (CARVALHO E FIGUEIREDO FILHO, 2014).

2.4.2. Estádio I

No início do carregamento, quando o momento fletor possui pequena intensidade comparada a capacidade resistiva da peça, as tensões de tração que surgem no concreto não ultrapassam a resistência a tração característica do material (f_{ctk}). Assim, não surgem fissuras no elemento estrutural (FIGURA 4). O diagrama de tensões apresenta comportamento linear, valendo-se a Lei de Hooke (PINHEIRO et al, 2007).

Figura 4 – Comportamento do concreto no Estádio I (flexão pura)



Fonte: Adaptado de Pinheiro et al (2007).

Para obtenção da inércia da seção de concreto armado no estádio I, a área de aço existente é transformada em uma área equivalente de concreto, e a inércia é calculada segundo sua definição clássica. Considera-se também que o concreto resiste aos esforços de tração neste estádio. Para calcular a inércia da seção no estádio I **(I_I)**, primeiramente é necessário calcular a posição da linha neutra da seção, utilizando a equação 17, conforme Carvalho e Figueiredo Filho (2014).

$$x_I = \frac{\frac{b \cdot h^2}{2} + (\alpha_e - 1) \cdot A_s \cdot d}{b \cdot h + (\alpha_e - 1) \cdot A_s} \qquad (15)$$

Sendo:

b = largura da seção;

h = Altura efetiva da seção;

A_s = Área de aço da seção;

d = Altura útil da seção;

α_e = Relação entre o módulo de deformação do aço e do concreto.

O valor de α_e é calculado através da razão:

$$\alpha_e = \frac{E_s}{E_{cs}} \tag{16}$$

Onde:

E_s = Módulo de elasticidade do aço.

E_{cs} = Módulo de deformação secante do concreto.

Por fim, é calculada a inércia da seção no estádio I utilizando a equação 17.

$$I_I = \frac{b \cdot h^3}{12} + b \cdot h \cdot \left(x_I - \frac{h}{2}\right)^2 + (\alpha_e - 1) \cdot A_s \cdot (d - x_I^2) \tag{17}$$

Onde:

b = largura da seção;

h = Altura da seção;

d = Altura efetiva da seção;

x_I = Posição da linha neutra da seção no estádio I;

A_s = Área de aço da seção;

α_e = Relação entre o módulo de deformação do aço e do concreto.

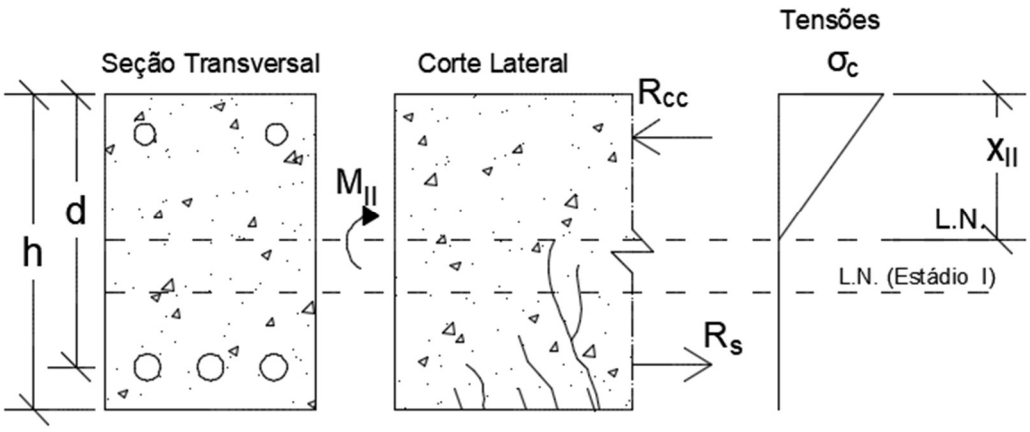
O final do estádio I ocorre quando o momento atuante na seção supera o momento de fissuração do concreto (M_r), onde há uma diminuição significante da sua rigidez, o que causa o aumento do estado de deformação e a redistribuição de cargas na estrutura.

2.4.3. Estádio II

Segundo Carvalho (1994), no estádio II surgem as primeiras fissuras no concreto e a sua resistência à tração é desconsiderada, logo todos os esforços desta natureza

são resistidos pelo aço. Assim o equilíbrio da seção ocorre ao igualar-se os esforços de compressão no concreto e na armadura comprimida com aqueles atuantes na armadura tracionada (FIGURA 5).

Figura 5 – Comportamento do concreto no Estádio II (flexão pura)



Fonte: Adaptado de Pinheiro et al (2007).

Assim, é necessário calcular novamente a posição da linha neutra da seção, para o novo equilíbrio de esforços. A equação a seguir apresenta duas raízes, sendo um destes valores o correspondente a posição da linha neutra da seção neste estádio de deformação (CARVALHO E FIGUEIREDO FILHO, 2014).

$$\frac{b}{2} \cdot x_{II}^2 + (\alpha_e - 1) \cdot A_s \cdot x_{II} - (\alpha_e - 1) \cdot A_s \cdot d = 0 \qquad (18)$$

Onde:

b = largura da seção;

d = Altura efetiva da seção;

x_{II} = Posição da linha neutra da seção no estádio II;

A_s = Área de aço da seção;

α_e = Relação entre o módulo de deformação do aço e do concreto.

Com a posição da linha neutra recalculada, pode-se estimar o valor do momento de inércia da seção fissurada de concreto, através da equação 19.

$$I_{II} = \frac{b \cdot {x_{II}}^3}{3} + \alpha_e \cdot A_s \cdot (d - x_{II})^2 \qquad (19)$$

Onde:

b = largura da seção;

d = Altura efetiva da seção;

x_{II} = Posição da linha neutra da seção no estádio II;

A_s = Área de aço da seção;

α_e = Relação entre o módulo de deformação do aço e do concreto.

2.4.4. Estádio III

Neste estádio de deformação, a carga resultante na seção é próxima ao valor máximo que ela pode resistir, onde inicia-se a plastificação do concreto comprimido e as tensões no aço podem superar o valor de escoamento. Para dimensionar uma estrutura capaz de resistir os esforços na sua ruptura, deve-se considerar este estádio de deformações (PINHEIRO et al, 2007).

Neste estádio, as tensões no concreto possuem o comportamento de um diagrama parábola-retângulo, apresentado na Figura 1. Entretanto a norma brasileira permite a consideração de um diagrama retangular, desde que haja garantia que a resultante da compressão e o braço de alavanca em relação a posição da linha neutra sejam próximos nos dois casos (ABNT, 2014).

2.4.5. Momento de Fissuração

O momento de fissuração do concreto (M_r), aquele que separa o comportamento do concreto nos estádios I e II, pode ser obtido pela equação 20, retirada da NBR 6118 (ABNT, 2014), item 17.3.1.

$$M_r = \alpha \cdot f_{ct} \cdot {I_c}/{y_t} \qquad (20)$$

Onde:

M_r = Momento de fissuração da peça;

α = Fator que correlaciona a resistência à tração na flexão com a resistência à tração direta, com valor 1,5 para seções retangulares;

f_{ct} = Resistência à tração direta do concreto; deve ser utilizado o valor $f_{ctk,inf}$ para o estado limite de formação de fissuras (ELS-F), e $f_{ct,m}$ no estado limite de deformação excessiva (ELS-DEF);

I_c = Momento de inércia da seção bruta de concreto;

y_t = Distância do centro de gravidade da seção à fibra mais tracionada;

Segundo Carvalho e Figueiredo Filho (2014), o controle da fissuração de elementos em concreto armado é indispensável para não comprometer a vida útil e garantir o bom funcionamento da estrutura. Assim, cabe ao projetista a escolha correta dos materiais utilizados e a verificação das tensões e deslocamentos, de acordo com os limites estabelecidos pela NBR 6118 (ABNT, 2014).

2.5. Rigidez à Flexão

De acordo com Silva (2012), a não linearidade física é uma propriedade intrínseca dos materiais, que provoca perda da rigidez dos elementos estruturais. O comportamento do concreto armado se deve, portanto, ao fato deste ser um material heterogêneo, que apresenta comportamento não-linear de tensão e deformação.

A rigidez de uma peça de concreto armado é influenciada por duas parcelas, o módulo de elasticidade do concreto com idade de 28 dias (E_{ci}), o módulo de elasticidade do aço das armaduras de flexão (E_s), e a inércia efetiva da seção (I_{ef}). Esta segunda parcela pode ser obtida em função dos momentos de inércia nos estádios I e II puro (CARVALHO, 1994).

Os equacionamentos mais utilizados para cálculo da inércia efetiva de uma seção fissurada de concreto, aqueles adotados pelas normas brasileira e norte americana, são apresentados a seguir.

2.5.1. Equação de Branson

Para realização de cálculo analítico das flechas em elementos estruturais considerando a fissuração do concreto, Branson (1968) propôs um modelo para calcular a inércia efetiva de uma viga, sabendo-se que a peça apresenta trechos com e sem fissuras. Esse modelo semiempírico apresenta ótimos resultados quando comparado a casos reais, sendo por isso adotado pela NBR 6118:2014 (Associação Brasileira de Normas Técnicas, 2014), e pela ACI 318-18 (American Concrete Institute, 2018) para o cálculo de deslocamento imediato em vigas fissuradas de concreto. A inércia efetiva de uma seção fissurada de concreto (I_{ef}), pode ser calculada através da equação 21, conforme Branson (1968).

$$I_{ef} = \left(\frac{M_r}{M_a}\right)^n \cdot I_I + \left[1 - \left(\frac{M_r}{M_a}\right)^n\right] \cdot I_{II} \qquad (23)$$

Onde:

I_{ef} = Momento de inércia efetivo;

M_r= Momento de fissuração da peça;

M_a= Momento atuante na seção mais solicitada da viga;

I_I = Momento de inércia da seção no estádio I;

I_{II} = Momento de inércia da seção no estádio II;

n = Índice de valor igual a 4, para situações em que a análise é feita em apenas uma seção da peça, ou igual a 3, quando se faz a análise da peça ao longo de todo seu comprimento.

2.5.2. Equação de Bischoff

Segundo Bischoff (2005), a equação proposta por Branson (1968) não funciona perfeitamente em todos os casos, pois esta superestima a rigidez das vigas de concreto armado, sendo necessário portanto a utilização de um fator de correção na obtenção das flechas através deste equacionamento. Movido pelas limitações da aplicação deste fator corretivo, o autor então propôs uma alternativa à equação de Branson, que segundo ele funciona para uma maior variedade de vigas em concreto armado, seja com aço ou plástico reforçado por fibras. Esse novo equacionamento foi incorporado pela ACI 318-

19 (American Concrete Institute, 2019). A inércia efetiva de Bischoff (2005) é calculada de acordo com a equação que segue.

$$I_{ef} = \frac{I_{II}}{1-\left(\frac{M_r}{M_a}\right)^2 \cdot \left[1-\left(\frac{I_{II}}{I_I}\right)\right]} \quad (22)$$

Onde:

I_{ef} = Momento de inércia efetivo;

M_r= Momento de fissuração da peça;

M_a= Momento atuante na seção mais solicitada da viga;

I_I = Momento de inércia da seção no estádio I;

I_{II} = Momento de inércia da seção no estádio II;

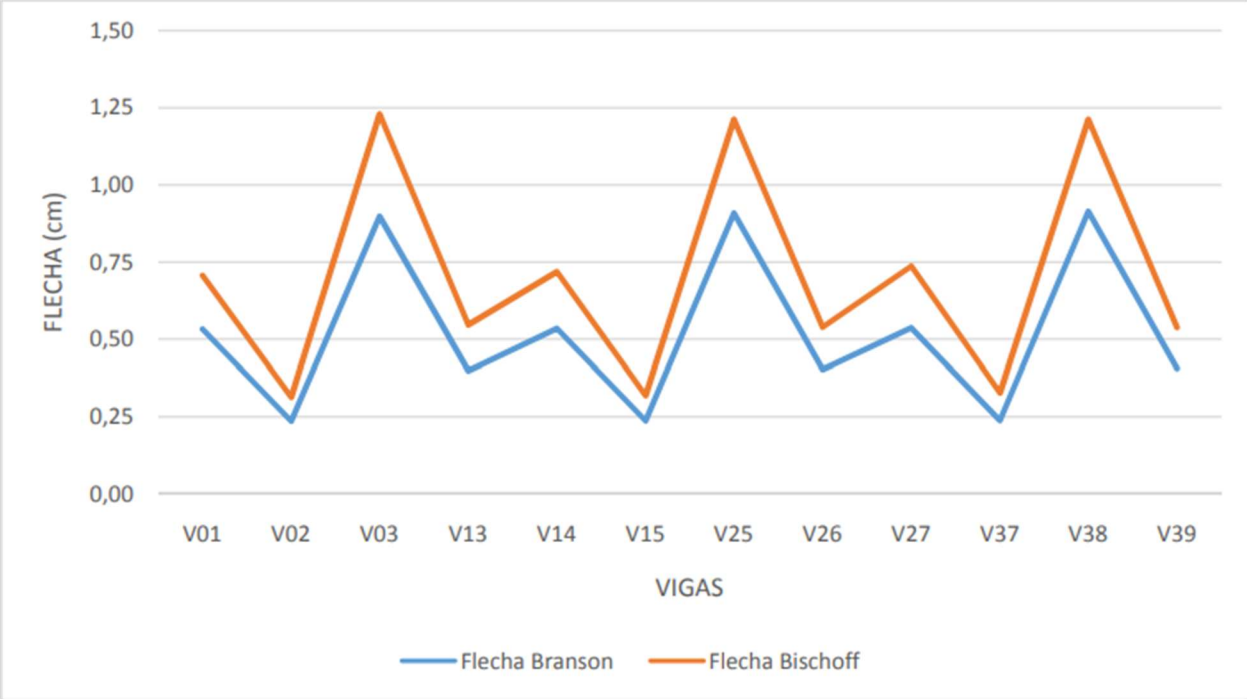
Em seu trabalho, Souza (2019) realizou um estudo comparativo entre os métodos de Branson e Bischoff para o cálculo da flecha imediata em vigas biapoiadas. A autora realizou um estudo paramétrico destas equações para cálculo da inércia efetiva de uma seção fissurada, calculando múltiplos casos de vigas biapoiadas de concreto, variando os parâmetros de entrada (TABELA 4). A autora organizou os dados obtidos também em gráficos, a fim de comparar o resultado obtido entre diferentes vigas para os dois métodos utilizados (FIGURA 4).

Tabela 4 – Dados das vigas e valor de flechas calculadas

VIGA	RESISTÊNCIA (Mpa)		SEÇÃO	VÃO (m)	AS (cm²)	Flecha (Branson) (cm)	Flecha (Bischoff) (cm)
V01	C20	20	20x60	6	10,05	0,53	0,71
V13	C20	20	20x60	4	10,05	0,24	0,31
V25	C20	20	20x40	6	6,03	0,90	1,23
V37	C20	20	20x40	4	6,03	0,40	0,55
V49	C30	30	20x60	6	15,08	0,54	0,72
V61	C30	30	20x60	4	15,08	0,24	0,32
V73	C30	30	20x40	6	9,05	0,91	1,21
V85	C30	30	20x40	4	9,05	0,40	0,54
V97	C40	40	20x60	6	20,11	0,54	0,74
V109	C40	40	20x60	4	20,11	0,24	0,33
V121	C40	40	20x40	6	12,07	0,91	1,21
V133	C40	40	20x40	4	12,07	0,41	0,54

Fonte: Adaptado de SOUZA (2019).

Figura 4 – Comparativo de flechas para diferentes vigas



Fonte: SOUZA (2019).

2.6. Análise Matricial de Estruturas

A análise matricial das estruturas é um método para obtenção dos esforços e deslocamentos que ocorrem em uma estrutura devido aos carregamentos nela aplicados. O método clássico considera elementos infinitesimais da estrutura, sendo possível exprimir o comportamento desta através da integração matemática. Por outro lado, o método dos elementos finitos considera que a estrutura possui uma quantidade limitada de elementos, com comprimento definido, e ligadas por pontos nodais. Após calcular as tensões e deformações que atuam em cada um destes pontos da estrutura, é possível expressar matematicamente o comportamento elástico e mecânico de cada elemento, substituindo a integralização matemática por um sistema de equações facilmente resolvidos matricialmente (MOREIRA, 1997).

Azevedo (2003) classificou os tipos de análise possíveis utilizando o método dos elementos finitos, de acordo com o tipo de estrutura considerada, a consideração de

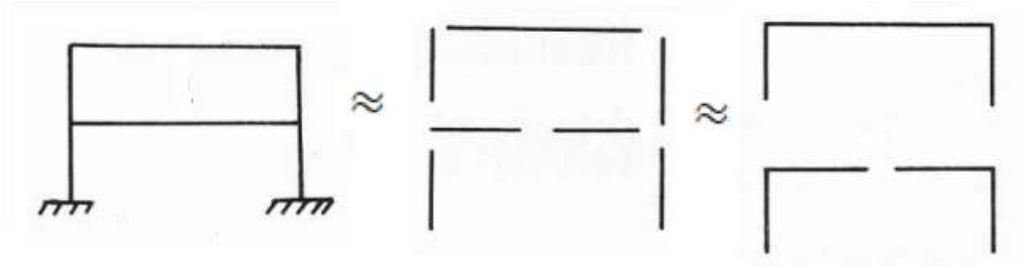
efeitos dinâmicos, e a consideração da perda de linearidade dos materiais, das geometrias e dos carregamentos. Pode-se, ainda, classificar a análise realizada quanto ao número de graus de liberdade considerados para cada ponto nodal dela.

2.6.1. Analogia de barras

O cálculo matricial das estruturas poderia ser considerado como um primeiro capítulo do método dos elementos finitos, aplicável a estruturas reticuladas, ou seja, constituídas por barras prismáticas com seção transversal de dimensões muito inferiores ao comprimento do elemento. Cada barra do modelo possui uma rigidez à flexão, valor que depende do momento de inércia da barra e do módulo de elasticidade do material (AZEVEDO, 2003).

Ao estudar estruturas reticuladas, os elementos que constituem esta podem ser representados por linhas que contenham o eixo da barra. Assim, um ponto nodal pode ser qualquer ponto sobre este eixo (MOREIRA, 1997).

Figura 6 – Exemplos de subdivisões de um pórtico em elementos finitos



Fonte: Adaptado de Moreira (1997).

2.6.1.1. Barras

Na Figura 6, a cada subdivisão da estrutura são formados novos elementos de barra, delimitados pelos nós de suas extremidades. As cargas atuantes nas barras são transformadas em reações nestes nós, e compatibilizadas entre as barras através das equações de equilíbrio estático e da compatibilidade de deformações (MOREIRA, 1997).

2.6.1.2. Nós

Entende-se por nó todos os pontos que formam a estrutura, onde são consideradas atuantes todas as cargas. Um nó é definido por suas coordenadas globais, ou seja, sua posição em relação a origem do sistema de coordenadas da estrutura. Em um espaço tridimensional, cada nó apresenta seis graus de liberdade, representando as três translações e três rotações possíveis para cada um destes nós.

No caso de viga não inclinadas, específicas para este trabalho, um nó pode ser definido apenas por sua posição em relação ao eixo longitudinal da estrutura. Considera-se que cada nó possui apenas três graus de liberdade, sendo duas translações e uma rotação, relacionada a curvatura da peça.

2.6.2. Equacionamento matricial

Dada uma viga contínua, constituída por *m* elementos de barra interligados por *n* nós, o vetor que contém os deslocamentos de um nó “i” qualquer é denotado pela equação 25.

$$\{u\}_i = \begin{Bmatrix} u_{xi} \\ u_{yi} \\ \varphi_i \end{Bmatrix} \tag{23}$$

Onde:

u_{xi} é o deslocamento em relação ao eixo global x do nó “i”;

u_{yi} é o deslocamento em relação ao eixo global y do nó “i”;

φ_i é a rotação do nó “i” em relação à sua orientação inicial.

De forma análoga, o vetor de deslocamentos nodais globais, de uma estrutura formada por “n” nós, é obtido pela junção dos vetores deslocamento de cada um dos nós, apresentado na equação 26.

$$\{U\} = \begin{Bmatrix} \{u\}_1 \\ \{u\}_2 \\ \{u\}_3 \\ \vdots \\ \{u\}_n \end{Bmatrix} = \begin{bmatrix} u_{x1} \\ u_{y1} \\ \varphi_1 \\ u_{x2} \\ u_{y2} \\ \varphi_2 \\ \vdots \\ u_{xn} \\ u_{yn} \\ \varphi_n \end{bmatrix} \quad (24)$$

Onde:

$\{u\}_n$ é o vetor de deslocamentos globais do nó “n”.

A matriz coluna {U}, resultante da equação 26, sempre terá sua maior dimensão igual ao número de nós da estrutura multiplicado pelo número de graus de liberdade de cada nó. Este vetor descreve apenas o movimento de corpo rígido de cada nó da estrutura. Para estudar o comportamento entre os nós, portanto, é necessário analisar as deformações de cada barra estrutural utilizando a equação 27, ilustrada pela Figura 7.

$$\{d\}_A = \begin{Bmatrix} r_i \\ r_j \\ \delta_A \end{Bmatrix} \quad (25)$$

Onde:

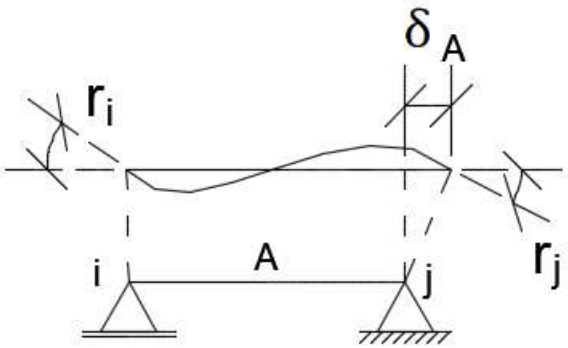
$\{d\}_A$ é o vetor de deslocamentos locais do elemento A.

r_i é a rotação do elemento em i, em relação ao eixo do elemento;

r_j é a rotação do elemento em j, em relação ao eixo do elemento;

δ_A é o alongamento ou encurtamento do elemento A.

Figura 7 – Deformações locais do elemento A entre os nós 1 e 2



Fonte: Adaptado de Moreira (1997).

2.6.3. Método da rigidez direta

O método da rigidez direta proporciona equacionamento matricial para obter os deslocamentos de uma estrutura hiperestática, utilizando a matriz de rigidez global da estrutura. Para solicitações normais, o valor dos elementos desta matriz depende basicamente do módulo de elasticidade do material utilizado, da inércia e da área da seção, e do comprimento do elemento. Entende-se por matriz de rigidez [K], a matriz de transformação linear, que multiplicada pelo vetor de deslocamentos resulta no vetor das ações a eles associadas (MOREIRA, 1997).

2.6.4. Cálculo Incremental

O cálculo incremental, descrito por Carvalho (1994), é um método para consideração da perda de linearidade do concreto armado utilizando análise matricial de estruturas. Neste processo, a carga total atuante na estrutura é dividida em “n” de etapas de carga, e calculam-se os deslocamentos nodais devidos a esta parcela do carregamento. Com estes valores de deslocamentos para um carregamento incremental, torna-se possível obter as ações em cada nó da estrutura, através do método da rigidez, a cada incremento de carga.

Após a resolução elástica da estrutura a ser estudada para cada etapa de carga, e antes de realizar a próxima etapa de carregamento, é necessário verificar se as tensões normais de tração no concreto superam os valores limites para a formação de fissuras

pois, caso surjam fissuras na peça, haverá perda de rigidez do elemento estrutural, como citado anteriormente neste capítulo.

Caso o momento atuante em qualquer ponto da estrutura supere o valor do momento de fissuração do elemento estrutural de concreto armado, torna-se necessário corrigir a matriz de rigidez da estrutura, para contemplar a perda de rigidez proveniente da fissuração. No caso deste trabalho, esta perda de rigidez foi estimada utilizando-se da equação de Branson (1968), com valor do expoente “n” igual a 4. Assim, cada elemento que apresentar situação de fissuração terá sua inércia efetiva alterada, causando também alteração na matriz de rigidez global da estrutura [K].

Recalcula-se a etapa atual com a rigidez corrigida para os elementos fissurados, para garantir a redistribuição dos esforços devido à mudança no estado de equilíbrio. Se após a redistribuição não surgirem novas fissuras, o roteiro de cálculo avança para a próxima etapa de carga, e o processo se repete.

O processo para o ciclo “i” de carga é demonstrado nas equações 28 e 29, de acordo com Carvalho (1994).

$$\{\Delta U\} = [K]^{-1} \cdot \{\Delta P\} \quad (26)$$

$$\{U\}_i = \{U\}_{i-1} + \{\Delta U\} \quad (27)$$

Onde:

$\{\Delta U\}$ é o vetor dos deslocamentos nodais devidos ao carregamento atuante na etapa “i”;

$[K]$ é a matriz de rigidez modificada de acordo com a etapa “i”;

$[K]^{-1}$ é a matriz [K] inversa;

$\{\Delta P\}$ é o vetor de cargas que correspondem ao carregamento na etapa “i”;

$\{U\}_i$ é o vetor que contém os deslocamentos acumulados dos nós, causados pela carga atuante até a etapa “i”.

$\{U\}_{i-1}$ é o mesmo do vetor $\{U\}_i$, desconsiderando a parcela de carga da etapa atual.

2.7. Implementação em Python

Python é uma linguagem de programação incrivelmente eficiente, no sentido de ser capaz de resolver problemas em poucas linhas de código, quando comparada a outras linguagens de programação. Um dos atrativos para muitos programadores escolherem a linguagem Python é a facilidade de leitura, facilitando na depuração, na ampliação e continuação de códigos criados (MATTHES, 2016).

Por ser um projeto de desenvolvimento de código aberto, conta com uma grande comunidade de desenvolvedores que disponibilizam bibliotecas para a comunidade utilizar, conforme os termos de licença de código aberto. A seguir são apresentadas as bibliotecas utilizadas no desenvolvimento deste trabalho.

2.7.1. AnaStruct

A biblioteca AnaStruct (VINK, 2018) é uma implementação em Python do método dos elementos finitos em 2D. Esta biblioteca foi utilizada para calcular os esforços e deslocamentos atuantes nos nós da estrutura. Para modelar uma estrutura com esta ferramenta computacional, é necessário fornecer dados sobre a estrutura reticulada a ser analisada, sendo estes: a posição de cada um dos nós que constitui a estrutura; as vinculações destes nós; o módulo de elasticidade do material que constitui a barra; A área e o momento de inércia de cada elemento de barra; e os carregamentos atuantes em cada barra.

Ainda, a biblioteca permite a obtenção dos resultados nodais do sistema estrutural resolvido, retornando o valor de F_x, F_y, T_y, u_x, u_y e φ_y para cada nó da estrutura, sendo

F_x = Força na direção “x”;

F_y = Força na direção “y”;

T_y = Força cortante;

u_x = Deslocamento na direção “x”;

u_y = Deslocamento na direção “y”;

φ_y = Ângulo de rotação em relação ao eixo da barra.

2.7.2. Pandas

Para realizar o armazenamento de todos estes dados, foi utilizada a biblioteca Pandas, uma biblioteca que facilita a manipulação de tabelas de dados, além de permitir a leitura e escrita de dados em arquivos em formatos CSV, um formato de arquivo de texto simples para armazenar informações de planilhas e tabelas (PANDAS DEVELOPMENT TEAM, 2022).

2.7.3. Tkinter

Ainda, na criação da interface a ser utilizada pelo usuário para entrada dos dados da viga, utilizou-se a biblioteca Tkinter, que é a biblioteca padrão do Python para desenvolvimento de interface gráfica do utilizador (PYTHON SOFTWARE FOUNDATION, 2022).

3. METODOLOGIA

A metodologia de desenvolvimento deste trabalho contemplou as seguintes etapas:

3.1. Determinação dos Métodos de Análise
- i. Método de Branson;
- ii. Método de Bischoff;
- iii. Método de elementos finitos utilizando analogia de barras e correção de inércia por Branson, a partir de incrementos de carga;

3.2. Definição de parâmetros a serem considerados

3.3. Etapas de desenvolvimento do software
- 3.3.1. Modelagem da estrutura
- 3.3.2. Implementação dos métodos de análise
- 3.3.3. Criação da interface gráfica
- 3.3.4. Depuração do código

Cada um destes tópicos foi apresentado a seguir.

3.1. Determinação dos Métodos de Análise

Inicialmente fez-se um levantamento dos métodos propostos na literatura para estimativa de deslocamentos em vigas de concreto armado, considerando a não linearidade física do concreto após sua fissuração, conforme descritos no capítulo 2. Para o escopo deste trabalho, foram considerados três métodos para determinação de deslocamentos. Estes são os descritos no capítulo 2.

3.2. Definição de parâmetros a serem considerados

Neste trabalho são estudadas apenas vigas biapoiadas não inclinadas. Assim, é possível modelar computacionalmente uma viga de 4 metros de vão com apenas dois nós, (0,0), (400,0), sendo que a unidade de medida utilizada pelo software é centímetros. Ainda, foram considerados fixos os deslocamentos vertical e horizontal nos apoios da viga.

Definiu-se que os valores de f_{ck} abrangidos seriam aqueles menores que 50 MPa, contemplando concretos do grupo I. O módulo de elasticidade do concreto é calculado ao definir-se o f_{ck} do material, através da equação (8). Considerou-se neste trabalho concreto formado por agregado de granito, e aço do tipo CA-50. A única geometria permitida para vigas é a retangular, assim a área e o momento de inércia são obtidos ao definir-se a base e altura da seção transversal. Por fim, o carregamento deve ser definido, isto é, a carga permanente e a carga de utilização (sobrecarga acidental). Somadas ao peso próprio, estas representam as três cargas distribuídas que atuam na estrutura da viga. Através de combinação de ações, é calculada a carga distribuída total a ser aplicada na estrutura.

Para os valores para o coeficiente de Poisson, peso específico do concreto armado, coeficiente de dilatação térmica e coeficiente de ponderação da resistência do concreto e do aço, foram utilizados os limites fornecidos pela NBR 6118 (Associação Brasileira de Normas Técnicas, 2014).

3.3. Etapas de desenvolvimento do software

3.3.1. Modelagem da estrutura

Definidos os métodos de cálculo a serem utilizados, a primeira parte de desenvolvimento do algoritmo foi a modelagem da estrutura em linguagem computacional, ou seja, a tradução dos dados de entrada em variáveis prontas para serem utilizadas em cada um dos casos citados no item anterior. Para isto, foi escolhida a linguagem Python, utilizando as bibliotecas descritas no capítulo 2.

Na analogia de barras, cada barra é definida por dois nós da estrutura. Ao mapear a posição destes nós, das informações de vinculação, das propriedades físicas e geométricas dos materiais constituintes da barra, e da carga a ser aplicada na estrutura, torna-se possível o cálculo programático para obtenção de esforços e deslocamentos em estruturas reticuladas.

3.3.2. Implementação dos métodos de análise

Para permitir a comparação efetiva entre cada um dos métodos revisados no capítulo 2, o paradigma de desenvolvimento escolhido foi criar um script que simula carregamento incremental em uma viga em concreto armado. Desta forma, a carga fornecida pelo usuário do aplicativo é na verdade a carga final a ser aplicada na estrutura, que será atingida através de incrementos no carregamento atuante. Com a estrutura modelada, o deslocamento da viga em cada etapa de carga pode ser obtido em função da matriz de rigidez da peça. Assim, tendo-se em mãos o valor da rigidez local de cada barra para cada um dos métodos de análise, é possível calcular simultaneamente o deslocamento esperado para os diferentes valores de inércia efetiva da seção.

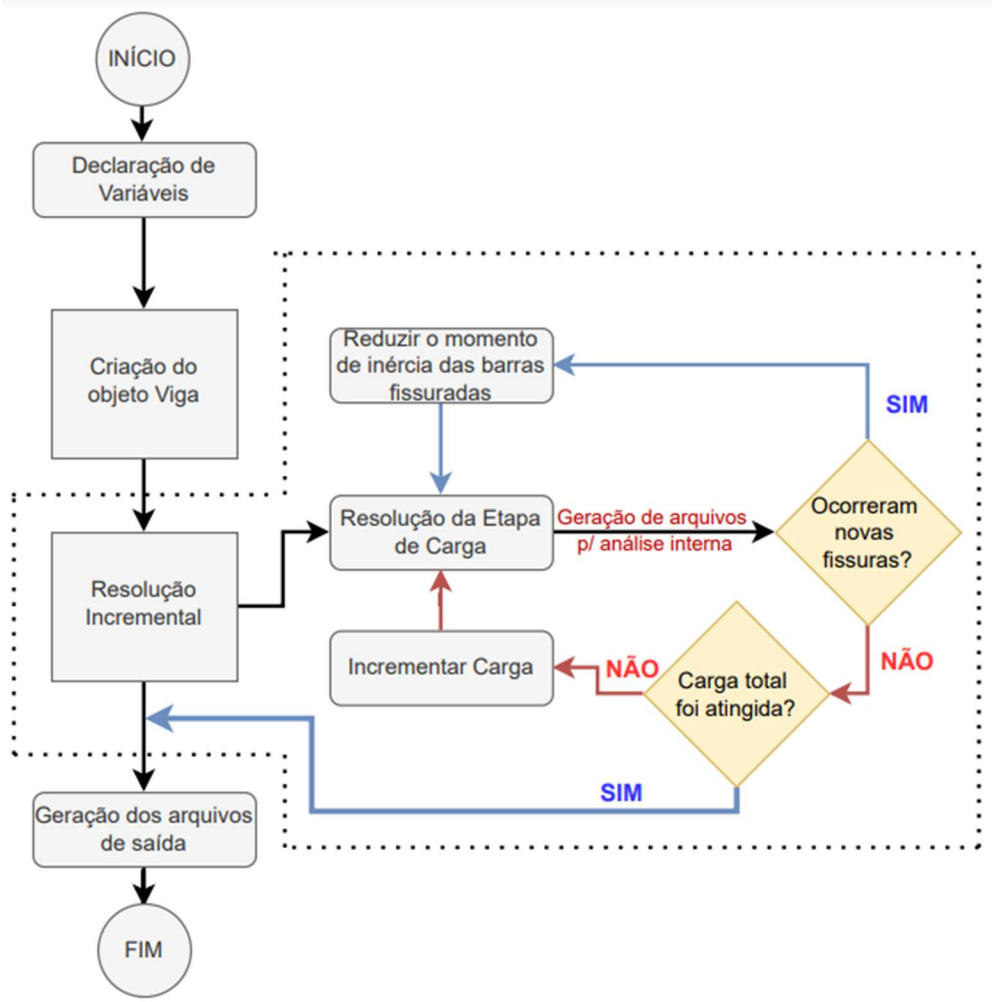
Na obtenção de deslocamentos em vigas utilizando o método de Branson (i) e o método de Bischoff (ii), conforme o equacionamento apresentado na literatura, o script verifica a cada etapa de carregamento se o momento atuante na seção, em qualquer ponto da estrutura, supera o momento de fissuração da peça. Em caso negativo, a rigidez da seção é mantida igual ao seu valor no estádio I puro. No caso positivo, o momento de inércia da peça inteira é substituído pelo valor obtido pela equação (8) oferecidas por estes métodos, e a flecha elástica é calculada para os dois equacionamentos.

Para o método dos elementos finitos utilizando analogia de barras com correção da inércia efetiva por Branson (iii), a cada ciclo de incremento de carga que o script realiza, a inércia dos elementos de barra que apresentam momento atuante superior ao momento de fissuração da peça é substituída pelo valor obtido na equação de Branson (1968).

A Figura 10 contém uma esquematização da rodagem do script desenvolvido para cálculo de deslocamentos utilizando método dos elementos finitos, com correção do momento de inércia a cada etapa de carga, utilizando a equação de Branson (iii). Em

seguida, cada uma das partes é detalhada para melhor compreensão do seu funcionamento.

Figura 8 – Fluxograma e rodagem do script para cálculo incremental



Fonte: Elaborado pelo autor

i. Declaração de variáveis

Nesta etapa deve-se definir os valores de entrada necessários para a rodagem do script, sendo estes:

Fck	Resistência característica do concreto;
Base	A dimensão da base da seção retangular;
Altura	A dimensão da altura da seção retangular;
As	Área de armadura positiva da seção;
As’	Área de armadura negativa da seção;
Cobrimento	O valor de cobrimento da seção;
Vão	Comprimento do primeiro vão da viga;
Qtd. Barras	Quantidade de elementos de barra a serem utilizados;
Carga total	O carregamento distribuído total que atua na viga;
Etapas de carga	Número de etapas a serem realizadas;

ii. Criação do objeto de Viga

Com os dados fornecidos, o script inicializa o objeto de viga. Este objeto armazena todos os dados necessários para o cálculo do deslocamento, sendo estes:

Viga: nome fictício dado ao objeto em Python. (Ex.: V001).

Nós: lista de todos os nós que compõem a viga; o número de nós depende da discretização utilizada para o cálculo da viga (Ex.: [(0,0), (1,0), (2,0), (3,0), (4,0), (5,0)]).

Suportes: lista de todos os nós da viga que representam os apoios. (Ex.: [(0,0), (5,0)])

Seção: a seção da viga é representada por outro objeto em Python, que possui os dados do material, geometria e armadura de flexão da viga.

Carga: a carga distribuída aplicada à viga.

Ao criar o objeto de viga, as propriedades do elemento estrutural são calculadas e armazenadas em variáveis para serem usadas durante o cálculo. O equacionamento utilizado para este cálculo é indicado no capítulo 2.

iii. Resolução incremental

Após criado o objeto, e com todos os valores anteriormente calculados, é iniciada a resolução incremental da viga. Esta é a função do script com o maior tempo de execução, onde são realizadas quantas iterações forem previamente determinadas para resolver a estrutura.

a. Resolução da primeira etapa de carga

O software então executa o cálculo linear para a primeira etapa de carga, utilizando a função *solve_beam()*. O programa também calcula para cada etapa de carga o valor de deslocamento utilizando as equações de Branson e Bischoff. A saída são arquivos em formato de tabelas, com resultados de esforços, deslocamentos e ângulo de rotação de cada um dos nós. Estes resultados são apresentados através de gráficos na própria aplicação

b. Verificação da fissuração

Com os valores obtidos anteriormente, é realizada comparação do momento atuante em cada elemento discreto com o momento de fissuração da peça.

- Caso o momento atuante seja maior que o momento de fissuração, a rigidez do elemento de barra é alterada e a etapa de carga atual é recalculada;
- Caso contrário, a carga é incrementada com o valor definido no início da rodagem.

c. Critério de parada

O script finaliza o cálculo quando a carga aplicada na viga corresponde ao valor fornecido pelo usuário na entrada de dados, sem verificar se a solicitação de cálculo supera o esforço resistente da peça.

iv. Geração de arquivos de saída (tabelas e gráficos).

As tabelas geradas durante o processo são então reutilizadas para obtenção dos deslocamentos máximos e mínimos da viga durante cada etapa de carga. Estas planilhas possuem também o valor de deslocamento calculado utilizando as inércias efetivas de Branson e de Bischoff. Assim, é possível gerar gráficos comparativos entre os três métodos utilizados.

v. Final do script

3.3.3. Criação da interface gráfica

Para facilitar a entrada de dados a serem calculados, foi criado um aplicativo para unir os scripts desenvolvidos, obtendo assim o deslocamento através de cada um dos métodos utilizados e salvando estes dados em arquivos de tabelas. A interface de usuário para este aplicativo foi dividida em três telas, a inicial onde são definidos os dados da viga, a tela de carregamento incremental da estrutura, e a tela para análise de resultados ou salvamento destes em arquivos de tabela.

4. RESULTADOS

Neste capítulo estão reunidos todos os resultados e exemplos de aplicação da ferramenta desenvolvida. O principal resultado é o aplicativo em si, que permite o estudo de vigas de concreto armado utilizando modelos de cálculo da norma brasileira e da norma norte americana, além de implementar o método dos elementos finitos com correção da rigidez pela equação de Branson. A seguir são apresentados exemplos de utilização do programa.

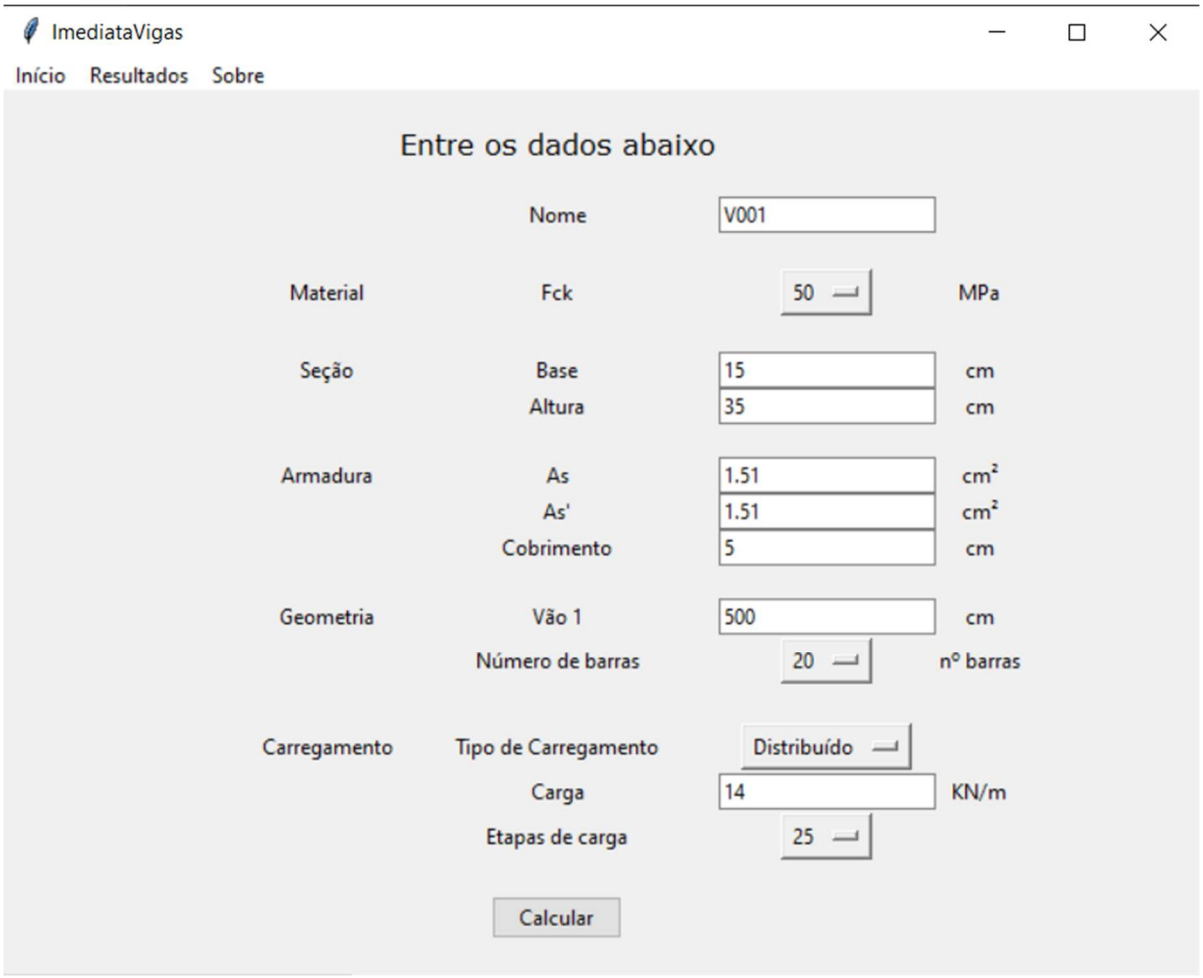
4.1. Execução do aplicativo

O aplicativo desenvolvido contém uma interface gráfica para o usuário, facilitando a entrada dos dados necessários para o cálculo da flecha imediata da viga desejada. A seguir são apresentadas imagens geradas pelo programa.

4.1.1. Abertura do programa

Ao iniciar, a tela para entrada de dados é apresentada, como mostra a Figura 11. O programa oferece valores iniciais para todos os campos, permitindo a utilização imediata. Foram estes dados padrões os utilizados na rodagem deste exemplo. Cabe ao usuário ajustar os parâmetros de forma correta para um bom funcionamento da aplicação desenvolvida. Se alguma das variáveis de entrada apresentar um valor que impossibilite a execução do script de cálculo, uma notificação é emitida para que o usuário corrija as variáveis pertinentes.

Figura 9 – Tela inicial da aplicação desenvolvida



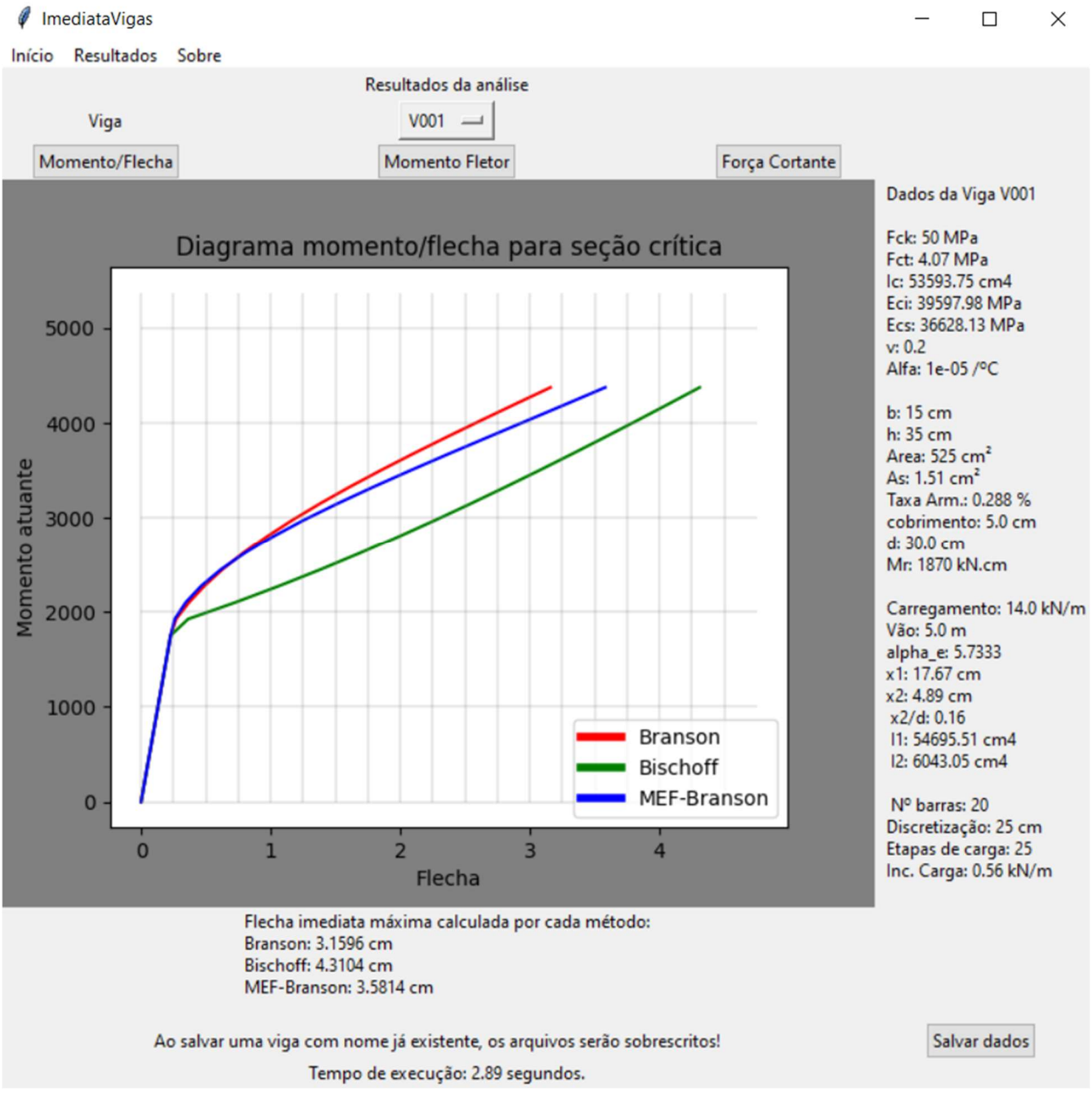
Fonte: Elaborada pelo autor.

4.1.2. Tela de resultados

Ao clicar no botão “calcular”, o programa realiza o cálculo dos deslocamentos utilizando os métodos citados no capítulo 3, e apresenta estes resultados em uma nova tela. Nesta, é possível visualizar o diagrama de momento por deformação do elemento de viga, para a seção crítica do elemento estrutural, com valores para cada etapa de carga. Ainda, são apresentados os dados calculados pelo programa durante sua rodagem, para análise e conferência das considerações feitas pelo programa. São apresentados também o valor máximo de deflexão obtido para cada um dos métodos, e

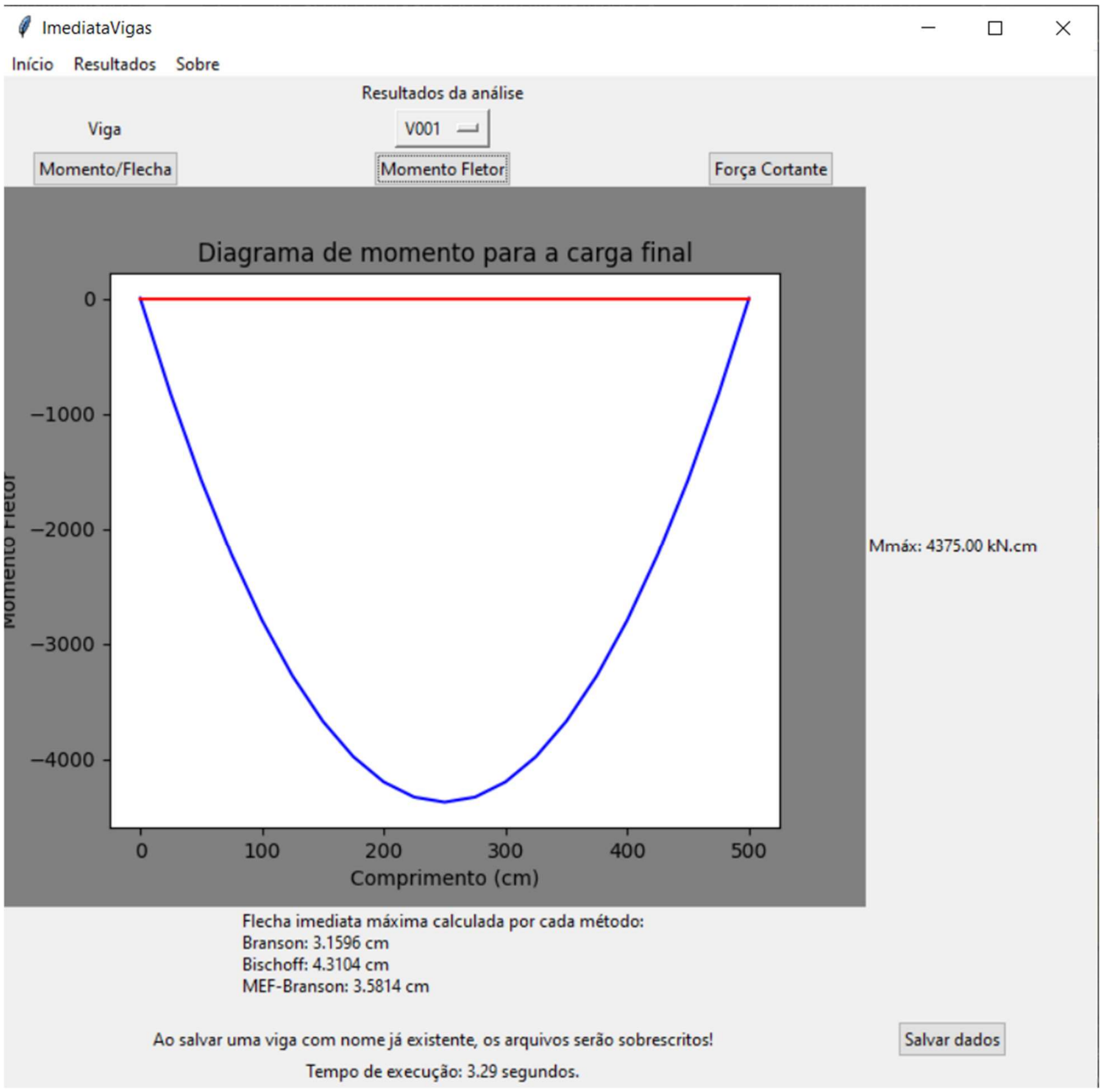
o valor máximo do momento fletor (FIGURA 12) e da força cortante(FIGURA 13). Nesta tela, também é possível guardar os resultados em arquivos de tabelas, ao clicar no botão salvar.

Figura 10 – Diagrama de momento atuante por flecha da seção crítica



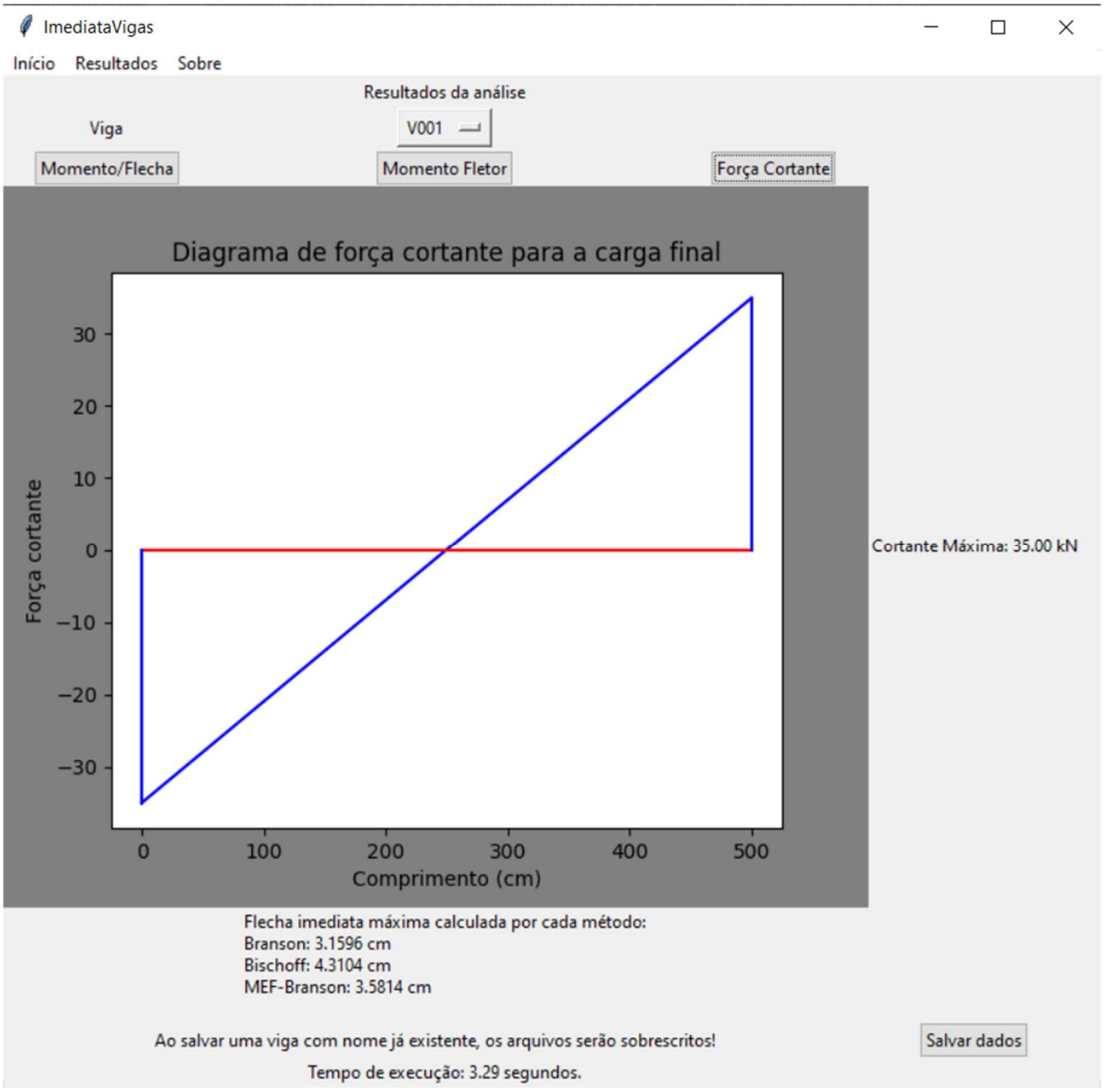
Fonte: Elaborada pelo autor.

Figura 11 – Diagrama de momento fletor para a última etapa de carga



Fonte: Elaborada pelo autor.

Figura 12 – Diagrama de força cortante para última etapa de carga realizada



Fonte: Elaborada pelo autor.

4.2. Impacto da quantidade de elementos finitos

Na tela inicial, é possível escolher a quantidade de barras da estrutura, e consequentemente, o número de nós a serem considerados no método dos elementos finitos. Utilizar valores muito grandes para esta variável pode acarretar um longo tempo

de processamento. Para exemplificar isto, uma viga foi calculada com os valores para número de barras disponíveis no programa, sendo estes 4, 10, 20, 50, 100 e 200 barras. Os dados de entrada para esta viga em particular foram: resistência característica do concreto de 35 MPa, base da seção 20 cm, altura de 40 cm, área de aço tracionado igual a 5 cm^2, área de aço comprimido igual a 1,51 cm^2 com 2,5 cm de cobrimento, e vão de 600 cm. A carga aplicada foi de 13 KN/m com 50 etapas de carregamento. Utilizando a biblioteca *time*, nativa do Python, foi possível calcular o tempo que o script levou para ser executado. A comparação entre os tempos de processamento obtidos e os valores de flechas máximas obtidas para cada um dos valores de número de barras disponíveis foram apresentados a seguir.

Tabela 5 – Resultados obtidos com variação do número de etapas de carga

	Número de barras					
	4	**10**	**20**	**50**	**100**	**200**
Branson	2,064	2,064	2,064	2,064	2,064	2,064
Bischoff	2,203	2,203	2,203	2,203	2,203	2,203
MEF-Branson	2,256	2,195	2,155	2,125	2,114	2,108
Tempo (s)	0,60	1,10	2,80	12,41	44,60	217,44

Fonte: Elaborada pelo autor.

Com esses resultados foi possível perceber que, quanto menos elementos de barra são utilizados no método dos elementos finitos, maior vai se tornando a diferença entre o deslocamento obtido pela equação de Branson com expoente n igual a 3, representado pela linha vermelha nas imagens anteriores, e o deslocamento obtido utilizando este mesmo expoente com valor igual a 4 com aplicação no método dos elementos finitos, representado em azul (MEF-Branson).

Ainda, ao analisar os tempos de execução, apresentados na última linha de resultados do programa, torna-se visível a significância do número de elementos finitos no aumento do tempo de execução do script. Recomenda-se, portanto, a utilização de no máximo 50 barras durante o cálculo para evitar tempos muito longos de execução.

4.3. Impacto da quantidade de etapas de carga

Bem como a quantidade de elementos que servem para modelar a viga a ser estudada, o número de etapas de carga a serem realizadas mostra-se relevante aos resultados obtidos. O impacto da variação desta variável foi analisado brevemente a seguir.

Definiu-se, mais uma vez, novos valores para as variáveis de entrada do script. Desta vez, utilizou-se resistência característica do concreto igual a 40 MPa, seção de 30x60 cm, área de aço igual a 15 cm^2 com cobrimento de 4 cm, vão de 600 cm, 20 barras de elementos finitos e carregamento total de 50 kN/m. Fixados estes valores, variou-se o número de etapas de carga a serem realizadas, e obteve-se os resultados de flecha imediata máxima, além do tempo de processamento (TABELA 5).

Tabela 6 – Resultados obtidos com variação do número de etapas de carga

	Número de etapas				
	5	**10**	**25**	**50**	**100**
Branson	1,419	1,419	1,419	1,419	1,419
Bischoff	1,484	1,484	1,484	1,484	1,484
MEF-Branson	1,474	1,474	1,474	1,474	1,474
Tempo (s)	0,55	1,00	2,47	4,90	9,76

Fonte: Elaborada pelo autor.

Nota-se que, ao varia a quantidade de etapas de carga, a flecha imediata máxima calculada não varia para nenhum dos métodos utilizados, pois caso haja fissuras no último estado de equilíbrio, a perda de rigidez das barras fissuradas causará o rebalanceamento do momento fletor, possivelmente gerando novas fissuras após recalcular a etapa de carga, garantindo a correção do momento de inércia para todos os elementos fissurados. É notável também que quanto maior o número de etapas, mais

suavizado será o comportamento do gráfico de momento por deformação, pois este apresentará mais pontos para o traçado da curva.

4.4. Limites de utilização

Além das variáveis que possuem o seu valor fixado no aplicativo, os demais valores são livres para serem alteradas pelo usuário, com a motivação da aplicação desenvolvida ser utilizada como ferramenta para o estudo de elementos estruturais. Desta forma é necessário atentar-se aos resultados obtidos, a fim de respeitar os limites estabelecidos pela norma brasileira no dimensionamento de vigas de concreto armado. Ainda, para efetiva aplicação dos resultados obtidos pelo programa em um caso real, é necessário calcular o efeito da fluência do concreto, que foi desconsiderado neste trabalho, além de realizar verificações para o estado limite último.

No processo de cálculo utilizado neste trabalho, não foi considerada a plastificação do elemento de concreto armado, ou seja, o comportamento da peça com escoamento do aço tracionado.

Finalmente, recomenda-se o uso de, pelo menos, 25 etapas de carga e 50 barras para o modelo de elementos finitos, visto que valores acima destes limites apresentam um grande aumento no tempo de processamento com pouca variação nos resultados obtidos.

5. CONCLUSÃO

O presente trabalho teve como foco o projeto e desenvolvimento de uma ferramenta computacional com interface gráfica para implementar o método dos elementos finitos utilizando a linguagem Python, resultando em um software de fácil utilização capaz de calcular o deslocamento imediato de vigas biapoiadas de concreto armado em situações de serviço. Através do equacionamento teórico revisado, foi possível utilizar simultaneamente três valores para o momento de inércia efetivo do elemento estrutural, permitindo a comparação dos deslocamentos encontrados para cada um destes modelos. O resultado é disponibilizado em gráficos e arquivos de tabelas. A verificação dos valores calculados com a utilização de tabelas e cálculos manuais foi também fundamental para verificar o correto funcionamento do aplicativo.

O aplicativo tem potencial de auxiliar no pré-dimensionamento de vigas, e na execução de laudos técnicos, facilitando a estimativa de momento atuante em uma viga, sabendo-se a sua deformação naquele momento.

O potencial deste aplicativo auxiliar em trabalhos futuros reside em sua utilização para análises paramétricas de vigas de concreto armado, ou para fomentar o desenvolvimento de outros softwares com maior estrutura, utilizando por exemplo banco de dados para salvar os resultados calculados, possibilitando a análise de inúmeras vigas simultaneamente. Outras possíveis expansões incluem a capacidade de calcular vigas contínuas, incluir análises considerando fases de carregamentos reais, a consideração dos efeitos de longo prazo, modelar o comportamento do concreto no estádio III, verificação dos estados limites, ou ainda modelagens mais robustas do elemento de viga, utilizando elementos sólidos ao invés da analogia de barras. Por último, é possível utilizar este estudo como uma introdução para o desenvolvimento de uma biblioteca para resolução de estruturas reticuladas, similarmente à AnaStruct (2018).

6. REFERÊNCIAS

AMERICAN CONCRETE INSTITUTE (2018). **ACI 318-18 Building Code Requirements for Structural Concrete: Commentary on Building Code Requirements for Structural Concrete (ACI 318R-18): an ACI Report**. American Concrete Institute. ACI, 2018.

AMERICAN CONCRETE INSTITUTE (2019). **ACI 318-19 Building Code Requirements for Structural Concrete: Commentary on Building Code Requirements for Structural Concrete (ACI 318R-19): an ACI Report**. American Concrete Institute. ACI, 2019.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (2014). **NBR 6118**: Projeto de Estruturas de Concreto – Procedimento. Rio de Janeiro: ABNT, 2014.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (2007). **NBR 7480:** Aço destinado a armaduras para estruturas de concreto armado– Especificação. Rio de Janeiro, 2007.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (2015). **NBR 8953:** Concreto para fins estruturais – Classificação pela massa específica, por grupos de resistência e consistência. Rio de Janeiro, 2015.

AZEVEDO, A. F. M. (2003). **Método dos Elementos Finitos**. 1ª Edição – Faculdade de Engenharia da Universidade do Porto. Portugal, 2003.

BISCHOFF P.H. (2005) **Reevaluation of deflection prediction for concrete beams reinforced with steel and fiber reinforced polymer**. Journal of Structural Engineering, v.131, i.5, p.752-767. 2005.

BRANSON, D. E. (1968). **Design procedures for computing deflections**. Journal of American Concrete Institute, v.65, i.9, p.730-742. 1968.

CARVALHO, R. C. (1994). **Análise não-linear de pavimentos de edifícios de concreto através da analogia de grelha**. 218p. Tese (Doutorado). Escola de Engenharia de São Carlos, Universidade de São Paulo, 1994.

CARVALHO, R. C., FIGUEIREDO FILHO, J. R. (2014) **Cálculo e detalhamento de estruturas usuais de concreto armado: segundo a NBR 6118:2014**. 4ª Edição – Editora da UFSCar. São Carlos, SP, 2014.

MATTHES, E. (2016). **Python Crash Course: a hands-on, project-based introduction to programming**. 1st Edition – No Starch Press, Inc. San Francisco, 2016.

MOREIRA, D. F. (1977). **Análise matricial das estruturas**. 1ª Edição – Livros Técnicos e Científicos Editora S.A.; Editora da Universidade de São Paulo. São Paulo, 1977.

OLUKUN, F. A. (1991). **Prediction of Concrete Tensile Strength from Its Compressive Strength: An Evaluation of Existing Relations for Normal Weight Concrete**. ACI Materials Journal. v.88, i.3, p. 302-309. 1991.

PANDAS DEVELOPMENT TEAM. (2022). **Pandas Documentation: 2022**. Página de documentação. Disponível em: < https://pandas.pydata.org/docs/>. Acesso em: 05 de set. de 2022.

PINHEIRO et. al. (2007). **Fundamento do Concreto e Projeto de Edifícios**. Apostila de ensino. 380p. Escola de Engenharia de São Carlos. São Carlos, SP. 2004.

PYTHON SOFTWARE FOUNDATION (2022). **Python Documentation: Graphical User Interfaces with Tk: 2022**. Página de documentação. Disponível em: <https://docs.python.org/3/library/tk.html/>. Acesso em: 05 de set. de 2022.

SILVA, A. E. F. (2012). **Contribuições ao estudo da não-linearidade física em vigas de concreto armado. 2012.** 175p. Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia Civil, Universidade Federal de Santa Maria, Santa Maria.

SOUZA, G. D. (2019). **Cálculo De Deslocamento em Vigas em Concreto Armado Considerando Não-Lineralidade Física Pela Formulação De Branson e Bischoff**. Trabalho de Conclusão de Curso. Universidade Federal de São Carlos, São Carlos, 2019.

VINK, R. (2018) **AnaStruct's Documentation**. Página de documentação. Disponível em: < https://anastruct.readthedocs.io/en/latest/>. Acesso em: 05 de set. de 2022.

APÊNDICE A – CÓDIGO DESENVOLVIDO EM PYTHON

standalone.py

```
from dataclasses import dataclass
from math import exp, ceil
import os
from shutil import rmtree
import time
import tkinter as tk
from tkinter import ttk

from anastruct import SystemElements, Vertex
import numpy as np
import pandas as pd

from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.lines import Line2D

LARGE_FONT = ("Verdana", 12)
RESOLUTION = '750x700'


# analysis.py

@dataclass
class BeamAnalysis:
    name: str
    cases: dict

    def get_graph_dataframe(self):
        df = pd.DataFrame()
        load_arr = [0, ]
        moment_arr = [0, ]
        branson_arr = [0, ]
        bischoff_arr = [0, ]
        mef_arr = [0, ]
        for c in self.cases.values():
            load_arr.append(c.load)
            moment_arr.append(c.get_max_moment())
            branson_arr.append(c.branson)
            bischoff_arr.append(c.bischoff)
            mef_arr.append(c.get_max_deflection())
        df['load'] = load_arr
        df['moment'] = moment_arr
        df['branson'] = branson_arr
        df['bischoff'] = bischoff_arr
        df['mef'] = mef_arr
        return df

    def get_bending_diagram_dataframe(self, total_length):
        df = pd.DataFrame()
        last_case_key = max([k for k in self.cases.keys()])
        last_case = self.cases[last_case_key]
```

```
        bar_length = total_length / len(last_case.bars.keys())
        len_arr = []
        moment_arr = []
        for v in last_case.bars.values():
            len_arr.append((v.nodes[0].id - 1) * bar_length)
            moment_arr.append(v.nodes[0].M)
            if v.id == max([key for key in last_case.bars.keys()]):
                len_arr.append((v.nodes[1].id - 1) * bar_length)
                moment_arr.append(v.nodes[1].M)
        df['length'] = len_arr
        df['moment'] = moment_arr
        return df

    def get_shear_diagram_dataframe(self, total_length):
        df = pd.DataFrame()
        last_case_key = max([k for k in self.cases.keys()])
        last_case = self.cases[last_case_key]
        bar_length = total_length / len(last_case.bars.keys())
        len_arr = []
        shear_arr = []
        for v in last_case.bars.values():
            len_arr.append((v.nodes[0].id - 1) * bar_length)
            shear_arr.append(v.nodes[0].V)
            if v.id == max([key for key in last_case.bars.keys()]):
                len_arr.append((v.nodes[1].id - 1) * bar_length)
                shear_arr.append(v.nodes[1].V)
        df['length'] = len_arr
        df['shear'] = shear_arr
        return df


@dataclass
class LoadCaseObject:
    load: float
    bars: dict
    branson: float
    bischoff: float

    def get_node_deflection(self, node_id):
        return

    def get_node_moment(self):
        pass

    def get_max_deflection(self):
        max_uy = 0
        for b in self.bars.values():
            for n in b.nodes.values():
                if n.uy > max_uy:
                    max_uy = n.uy
        return max_uy

    def get_max_moment(self):
        max_moment = 0
        for b in self.bars.values():
            for n in b.nodes.values():
                if abs(n.M) > max_moment:
```

```
                    max_moment = abs(n.M)
        return max_moment


@dataclass
class NodeObject:
    id: int
    V: float
    M: float
    uy: float
    phi: float


@dataclass
class BarObject:
    id: int
    EI: float
    EA: float
    cracked: bool
    nodes: dict

    def get_max_deflection(self):
        return max([n.uy for n in self.nodes.values()])

    def get_max_shear(self):
        return max([n.uy for n in self.nodes.values()])


# materials.py

@dataclass
class Rebar:
    fyk: int
    ys: float
    gamma: int
    Es: int
    As: int
    As_neg: int


@dataclass
class Concrete:
    fck: int
    yc: float
    gamma: int
    CP: int
    v: float = 0.20
    alpha: float = 0.00001

    def fcd(self):
        return self.fckj() / self.yc

    def fctm(self):
        return 0.3 * self.fckj() ** (2 / 3)

    def fctk_inf(self):
        return 0.7 * self.fctm()
```

```
    def fctk_sup(self):
        return 1, 3 * self.fctm()

    def eci(self):
        return 5600 * (self.fckj() ** (1 / 2))

    def ecs(self):
        return (0.8 + 0.2 * (self.fckj() / 80)) * self.ecij()

    def gc(self):
        return 0.4 * self.ecs()

    def fckj(self, days=28):
        """
        s = 0,38 para concreto de cimento CPIII e CPIV;
        s = 0,25 para concreto de cimento CPI e CPII
        s = 0,20 para concreto de cimento CPV-ARI; fonte NBR 6118:2014
        :param days: int
        :return: int
        """
        if self.CP < 1 or self.CP > 5:
            return -1
        elif self.CP < 3:
            s = 0.25
        elif self.CP < 5:
            s = 0.38
        else:
            s = 0.20
        betta = exp(s * (1 - (28 / days) ** 0.5))
        fckj = betta * self.fck
        return fckj

    def fcdj(self, j=28):
        return self.fckj(j) / self.yc

    def ecij(self, days=28):
        return 5600 * (self.fckj(days) ** (1 / 2))


@dataclass
class ReinforcedConcrete:
    concrete: Concrete
    rebar: Rebar


# sections.py

@dataclass
class RectangularSection:
    base: int
    height: int
    alpha: int = 1.5  # ABNT NBR 6118: 2014, art. 17.3, p. 17.3.1

    def area(self):
        return self.base * self.height
```

```
    def area_effective(self):
        return (self.base * self.height) / 1.5  # todo: test value

    def inertia(self):
        return (self.base * self.height ** 3) / 12


@dataclass
class ReinforcedConcreteSection:
    material: ReinforcedConcrete
    geometry: RectangularSection
    cover: int

    def d(self):
        return self.geometry.height - self.cover

    def alpha_e(self):
        return self.material.rebar.Es / self.material.concrete.ecs()

    def mcr(self):
        return (self.geometry.alpha * (self.material.concrete.fctm() / 10) *
self.geometry.inertia()) / \
               (self.geometry.height / 2)

    def x1(self):
        return (((self.geometry.base * self.geometry.height ** 2) / 2) + (
                self.alpha_e() - 1) * self.material.rebar.As * self.d()) / \
               (self.geometry.area() + (self.alpha_e() - 1) *
self.material.rebar.As)

    def inertia1(self):
        return self.geometry.inertia() + self.geometry.area() * (self.x1() -
self.geometry.height / 2) ** 2 + \
               (self.alpha_e() - 1) * self.material.rebar.As * (self.d() -
self.x1()) ** 2

    def x2(self):
        a = self.geometry.base / 2
        b = (self.alpha_e() - 1) * self.material.rebar.As
        c = - (self.alpha_e() - 1) * self.material.rebar.As * self.d()
        coeff = [a, b, c]
        possible_values = np.roots(coeff)
        for val in possible_values:
            if 0 < val < self.geometry.height:
                return val
        return 0

    def inertia2(self):
        return (self.geometry.base * self.x2() ** 3) / 3 + \
               self.alpha_e() * self.material.rebar.As * (self.d() -
self.x2()) ** 2

    def ea(self):
        return self.material.concrete.eci() * self.geometry.area()

    def ei1(self):
        return self.material.concrete.ecs() * self.inertia1()
```

```
# writer.py

def list_files(path):
    r = []
    for root, dirs, files in os.walk(path):
        for name in files:
            r.append(os.path.join(root, name))
    return r


def list_folders(path):
    r = []
    for root, dirs, files in os.walk(path):
        for name in dirs:
            r.append(os.path.join(root, name))
    return r


def get_max_deflections_output_df(output_path, beam_name):
    for folder in list_folders(output_path):
        print(folder)
        if folder == output_path + beam_name:
            print("yes!")
            max_displacement_df = pd.DataFrame()
            min_displacement_df = pd.DataFrame()
            for file in list_files(folder):
                df = pd.read_csv(file)
                maximum_displacement_row = df.loc[df['uy'].idxmax()]
                maximum_displacement_row['name'] = os.path.basename(file)
                max_displacement_df =
max_displacement_df.append(maximum_displacement_row, ignore_index=True)

                minimum_displacement_row = df.loc[df['uy'].idxmin()]
                minimum_displacement_row['name'] = os.path.basename(file)
                min_displacement_df =
min_displacement_df.append(minimum_displacement_row, ignore_index=True)
            for ele, row in max_displacement_df.iterrows():
                if row['cracked'] == 0:
                    max_displacement_df.at[ele, 'branson'] = row['uy']
                    max_displacement_df.at[ele, 'bischoff'] = row['uy']
            for ele, row in min_displacement_df.iterrows():
                if row['cracked'] == 0:
                    min_displacement_df.at[ele, 'branson'] = row['uy']
                    min_displacement_df.at[ele, 'bischoff'] = row['uy']
            return max_displacement_df, min_displacement_df
    return None


def write_outputs_to_csv(folder):
    max_displacement_df = pd.DataFrame()
    min_displacement_df = pd.DataFrame()
    for file in list_files(folder):
        df = pd.read_csv(file)
        maximum_displacement_row = df.loc[df['uy'].idxmax()]
        maximum_displacement_row.loc['name'] = os.path.basename(file)
```

```
        maximum_displacement_row.loc['q_load'] =
os.path.basename(file).split('-')[-1][:-4]
        max_displacement_df =
max_displacement_df.append(maximum_displacement_row, ignore_index=True)
        minimum_displacement_row = df.loc[df['uy'].idxmin()]
        minimum_displacement_row.loc['name'] = os.path.basename(file)
        minimum_displacement_row.loc['q_load'] =
os.path.basename(file).split('-')[-1][:-4]
        min_displacement_df =
min_displacement_df.append(minimum_displacement_row, ignore_index=True)
    max_displacement_df.sort_values('q_load', axis=0, ascending=True,
inplace=True)
    max_displacement_df.to_csv(folder + "../deslocamento_maximo.csv")
    min_displacement_df.sort_values('q_load', axis=0, ascending=True,
inplace=True)
    min_displacement_df.to_csv(folder + "../deslocamento_minimo.csv")


# beam.py


OUTPUT_PATH = 'análise/'
BEAM_DATA_FILE = 'output_analysis/beams_input_data.csv'


class Beam:
    def __init__(self,
                 name: str,
                 nodes: list,
                 sups: list,
                 section: ReinforcedConcreteSection,
                 q_load: float,
                 discretization: int,
                 load_step: float,
                 elements: dict = None,
                 data: pd.DataFrame = None,
                 analysis: BeamAnalysis = None):
        self.name = name
        self.nodes = nodes
        self.supports = sups
        self.section = section
        self.q_load = q_load
        self.discretization = discretization
        self.load_step = load_step
        self.ss = SystemElements()
        self.total_length = np.linalg.norm(np.array(nodes[-1].coords) - [0.0,
0.0])
        self.solved = False
        self.branson_deflection = 0
        self.bischoff_deflection = 0
        self.analysis = analysis
        if elements is None:
            elements = {}
        self.elements = elements
        if data is None:
            data = pd.DataFrame()
        self.data = data
```

```
    def save_analysis_data(self):
        dataframes = []
        for case in self.analysis.cases.values():
            df = pd.DataFrame()
            if not self.solved:
                return dataframes
            df['element'] = [x for x in case.bars.keys()]
            bars = case.bars.values()
            df['ei'] = [el.EI for el in bars]
            df['ea'] = [el.EA for el in bars]
            df['shear'] = [el.nodes[0].V for el in bars]
            df['shear_2'] = [el.nodes[1].V for el in bars]
            df['moment'] = [el.nodes[0].M for el in bars]
            df['moment_2'] = [el.nodes[1].M for el in bars]
            df['uy'] = [el.nodes[0].uy * 10 for el in bars]
            df['uy_2'] = [el.nodes[1].uy * 10 for el in bars]
            df['phi'] = [el.nodes[0].phi for el in bars]
            df['phi_2'] = [el.nodes[1].phi for el in bars]
            df['cracked'] = [x in self.cracked_elements().keys() for x in
df['element']]
            df.to_csv("vigas/V001/etapas/" + self.name + '-
{:.2f}.csv'.format(case.load * 100), index=False)
            dataframes.append(df)
        write_outputs_to_csv("vigas/V001/")
        return dataframes

    def write_analysis_to_csv(self, folder=""):
        self.get_analysis_dataframe().to_csv(OUTPUT_PATH + folder + self.name
+ '.csv', index=False)

    def branson_inertia(self, actual_moment):
        ief = branson_equation(abs(self.section.mcr()),
                               abs(actual_moment),
                               self.section.inertia1(),
                               self.section.inertia2(),
                               n=4)
        return ief

    def ei_br(self, actual_moment):
        effective_stiffness = self.branson_inertia(actual_moment) *
self.section.material.concrete.ecs()
        return effective_stiffness

    def add_elements(self):
        for ele in self.nodes:
            if ele == Node(0, 0):
                continue
            self.ss.add_element(location=[ele.coords], EA=self.section.ea(),
EI=self.section.ei1())
        for sup in self.supports:
            n = self.ss.find_node_id(Vertex(sup.node.coords))
            if not any(self.ss.supports_fixed):
                self.ss.add_support_hinged(node_id=n)
                continue
            self.ss.add_support_roll(node_id=n)
```

```
    def add_elements_from_dict(self):
        for k, v in self.elements.items():
            if k == 0:
                continue
            self.ss.add_element(location=[v.coords()], EA=v.EA, EI=v.EI)
        for sup in self.supports:
            if not any(self.ss.supports_fixed):
                n = self.ss.find_node_id(Vertex(sup.node.coords))
                self.ss.add_support_hinged(node_id=n)
                continue
            n = self.ss.find_node_id(Vertex(sup.node.coords))
            self.ss.add_support_roll(node_id=n)

    def cracked_elements(self):
        elements = {}
        if not self.solved:
            return elements
        for k, v in self.ss.element_map.items():
            if (abs(v.bending_moment[0]) > abs(self.section.mcr())) or (
                    abs(v.bending_moment[-1]) > abs(self.section.mcr())):
                start_node = Node(*v.vertex_1.coordinates)
                end_node = Node(*v.vertex_2.coordinates)
                moment_in_act = max(abs(v.bending_moment[0]),
abs(v.bending_moment[-1]))
                elements[k] = Bar(start_node, end_node,
int(self.section.ea()),
                                  ceil(self.ei_br(moment_in_act) / 100) *
100)
        return elements

    def get_max_deflection_value(self):
        if not self.solved:
            return -1
        df = self.get_analysis_dataframe()
        deflection_value_series = df.loc[abs(df['uy']) == max(abs(df['uy'])),
'uy']
        deflection_value = deflection_value_series.to_list()[-1]
        return deflection_value

    def solve_beam(self, load=0):
        if load == 0:
            load = self.q_load
        for k in self.ss.element_map.keys():
            self.ss.q_load(q=load, element_id=k)
        self.ss.solve()
        self.solved = True
        return self.solved

    def create_analysis_bars_and_nodes(self, load_case_object):
        df = self.get_analysis_dataframe()
        for ele, row in df.iterrows():
            new_node_1 = NodeObject(
                int(row['element']),
                float(row['shear']),
                float(row['moment']),
                float(row['uy']),
                float(row['phi'])
```

```
            )
            new_node_2 = NodeObject(
                int(row['element']) + 1,
                float(row['shear_2']),
                float(row['moment_2']),
                float(row['uy_2']),
                float(row['phi_2'])
            )
            new_nodes = {0: new_node_1, 1: new_node_2}

            new_bar = BarObject(row['element'], row['ei'], row['ea'],
row['cracked'], new_nodes)
            load_case_object.bars[new_bar.id] = new_bar
        return load_case_object

    def input_describe(self):
        description = "  Dados da Viga {}\n\n".format(self.name) + \
                      "  Fck: {}
MPa\n".format(self.section.material.concrete.fck) + \
                      "  Fct: {:.2f}
MPa\n".format(self.section.material.concrete.fctm()) + \
                      "  Ic: {:.2f}
cm4\n".format(self.section.geometry.inertia()) + \
                      "  Eci: {:.2f}
MPa\n".format(self.section.material.concrete.eci()) + \
                      "  Ecs: {:.2f}
MPa\n".format(self.section.material.concrete.ecs()) + \
                      "  v: {} \n".format(self.section.material.concrete.v) +
\
                      "  Alfa: {}
/°C\n\n".format(self.section.material.concrete.alpha) + \
                      "  b: {} cm\n".format(self.section.geometry.base) + \
                      "  h: {} cm\n".format(self.section.geometry.height) + \
                      "  Area: {} cm²\n".format(self.section.geometry.area())
+ \
                      "  As: {} cm²\n".format(self.section.material.rebar.As)
+ \
                      "  Taxa Arm.: {:.3f}
%\n".format(self.section.material.rebar.As /

self.section.geometry.area() * 100) + \
                      "  cobrimento: {} cm\n".format(self.section.cover) + \
                      "  d: {} cm\n".format(self.section.geometry.height -
self.section.cover) + \
                      "  Mr: {} kN.cm\n\n".format(int(self.section.mcr())) +
\
                      "  Carregamento: {:.1f} kN/m\n".format(abs(self.q_load)
* 100) + \
                      "  Vão: {:.1f} m\n".format(self.total_length / 100) + \
                      "  alpha_e: {:.4f} \n".format(self.section.alpha_e()) +
\
                      "  x1: {:.2f} cm\n".format(self.section.x1()) + \
                      "  x2: {:.2f} cm\n ".format(self.section.x2()) + \
                      "  x2/d: {:.2f}\n ".format(self.section.x2() /
self.section.d()) + \
                      "  I1: {:.2f} cm4\n ".format(self.section.inertia1()) +
\
```

```
                      "  I2: {:.2f} cm4\n\n ".format(self.section.inertia2()) \
                      "  N° barras: {:.0f}\n".format(self.total_length / self.discretization) + \
                      "  Discretização: {} cm\n".format(self.discretization) + \
                      "  Etapas de carga: {:.0f}\n".format(abs(self.q_load * 100) / self.load_step) + \
                      "  Inc. Carga: {} kN/m\n".format(self.load_step)
        return description

    def get_analysis_dataframe(self):
        df = pd.DataFrame()
        if not self.solved:
            return df
        df['element'] = [x for x in self.ss.element_map.keys()]
        values = self.ss.element_map.values()
        df['ei'] = [el.EI for el in values]
        df['ea'] = [el.EA for el in values]
        df['shear'] = [el.shear_force[0] for el in values]
        df['shear_2'] = [el.shear_force[-1] for el in values]
        df['moment'] = [el.bending_moment[0] for el in values]
        df['moment_2'] = [el.bending_moment[-1] for el in values]
        df['uy'] = [el.node_1.uz * 10 for el in values]
        df['uy_2'] = [el.node_2.uz for el in values]
        df['phi'] = [el.node_1.phi_y for el in values]
        df['phi_2'] = [el.node_2.phi_y for el in values]
        df['cracked'] = [x in self.cracked_elements().keys() for x in df['element']]
        return df


# deflections.py

def get_branson_deflection(length, ecs, mr, ma, inertia_1, inertia_2):
    return (5 * ma * length ** 2) / (48 * (ecs / 10) * branson_equation(mr, ma, inertia_1, inertia_2))


def branson_equation(mr, ma, inertia_1, inertia_2, n=3):
    effective_inertia = (mr / ma) ** n * inertia_1 + (1 - (mr / ma) ** n) * inertia_2
    return effective_inertia


def get_bischoff_deflection(length, ecs, mr, ma, inertia_1, inertia_2):
    return (5 * ma * length ** 2) / (48 * (ecs / 10) * bischoff_equation(mr, ma, inertia_1, inertia_2))


def bischoff_equation(mr, ma, inertia_1, inertia_2):
    effective_inertia = inertia_2 / (1 - (((mr / ma) ** 2) * (1 - (inertia_2 / inertia_1))))
    return effective_inertia


# elements.py
```

```
class Node:
    def __init__(self, xx, yy):
        self.xx = xx
        self.yy = yy
        self.coords = np.array([xx, yy])

    def __str__(self):
        return f"%d,%d" % (self.xx, self.yy)

    def __eq__(self, other):
        if isinstance(other, Node):
            return self.xx == other.xx and self.yy == other.yy


@dataclass
class Bar:
    node_a: Node
    node_b: Node
    EA: int
    EI: int

    def length(self):
        return np.linalg.norm(self.node_a.coords - self.node_b.coords)

    def coords(self):
        return self.node_b.coords


@dataclass
class Support:
    node: Node
    xx: bool
    yy: bool
    zz: bool


# solver.py

def solve_beam_incrementally(original_beam, load_step):
    beam_analysis = BeamAnalysis(original_beam.name, {})
    if original_beam.q_load < 0:
        load_step *= -0.01
    load = load_step
    already_cracked_elements = {}
    beam_elements = original_beam.elements
    beam = None
    new_name = ""
    while abs(load) <= round(abs(original_beam.q_load + load_step / 2), 4):
        new_load_case = LoadCaseObject(abs(load), {}, 0.0, 0.0)
        if new_name == original_beam.name + "L" + str(load):
            new_name = new_name + "+1"
        else:
            new_name = original_beam.name + "L{:.5f}".format(load)
        beam = Beam(new_name,
                    original_beam.nodes,
                    original_beam.supports,
```

```
                    original_beam.section,
                    load,
                    original_beam.discretization,
                    original_beam.load_step,
                    beam_elements,
                    original_beam.data)
        beam.add_elements_from_dict()
        beam.solve_beam()
        ma = abs(beam.q_load * beam.total_length ** 2) / 8
        if (ma != 0) and (ma > beam.section.mcr()):
            beam.branson_deflection =
get_branson_deflection(beam.total_length,
beam.section.material.concrete.ecs(),

abs(beam.section.mcr()), abs(ma),

abs(beam.section.inertia1()), abs(beam.section.inertia2()))
            beam.bischoff_deflection =
get_bischoff_deflection(beam.total_length,
beam.section.material.concrete.ecs(),

abs(beam.section.mcr()), abs(ma),

abs(beam.section.inertia1()), abs(beam.section.inertia2()

))
            new_load_case.branson = beam.branson_deflection
            new_load_case.bischoff = beam.bischoff_deflection
        else:
            beam.branson_deflection = beam.get_max_deflection_value()
            beam.bischoff_deflection = beam.get_max_deflection_value()
            new_load_case.branson = beam.branson_deflection
            new_load_case.bischoff = beam.bischoff_deflection
        new_load_case = beam.create_analysis_bars_and_nodes(new_load_case)
        beam_analysis.cases[new_load_case.load] = new_load_case
        new_cracked_elements = beam.cracked_elements()
        if new_cracked_elements != {}:
            if already_cracked_elements == {}:
                already_cracked_elements = new_cracked_elements
            elif new_cracked_elements == already_cracked_elements:
                load += load_step
            elif len(new_cracked_elements) < len(already_cracked_elements):
                load += load_step
            else:
                if new_cracked_elements.keys() ==
already_cracked_elements.keys():
                    step = True
                    for k in new_cracked_elements.keys():
                        if new_cracked_elements[k].EI ==
already_cracked_elements[k].EI:
                            continue
                        else:
                            if new_cracked_elements[k].EI <
already_cracked_elements[k].EI:
                                step = False
                    if step:
                        load += load_step
```

```
                already_cracked_elements.update(new_cracked_elements)
        else:
            load += load_step
        beam_elements.update(already_cracked_elements)
    if beam is not None:
        beam.name = original_beam.name
        beam.analysis = beam_analysis
        beam.solved = True
        return beam
    return None


# runner.py

def create_beam_from_dict(input_dict, discretization, load_step):
    section_geometry = RectangularSection(input_dict['b'], input_dict['h'])
    section_material = ReinforcedConcrete(Concrete(input_dict['fck'],
input_dict['yc'], input_dict['gamma'], 1),
                                          Rebar(500, 1.15, 7850, 210000,
input_dict['as'], input_dict["asl"]), )
    section = ReinforcedConcreteSection(section_material, section_geometry,
input_dict['cover'])
    total_len = input_dict['l1'] + input_dict['l2']
    x = 0
    last_node = Node(0, 0)
    node_dict = {"0,0": last_node}
    nodes = [last_node, ]
    length_step = discretization
    elements_dict = {}
    while x < total_len:
        x += length_step
        new_node = Node(x, 0)
        elements_dict[x / length_step] = Bar(last_node, new_node,
section.ea(), section.ei1())
        nodes.append(new_node)
        node_dict[str(new_node)] = new_node
        last_node = new_node
    if int(input_dict['l2']) == 0:
        supports = [Support(node_dict['0,0'], True, True, False),
                    Support(node_dict[f'%d,0' % input_dict['l1']], True,
True, False)]
    else:
        supports = [Support(node_dict['0,0'], True, True, False),
                    Support(node_dict[f'%d,0' % input_dict['l1']], True,
True, False),
                    Support(node_dict[f'%d,0' % int(input_dict['l2'] +
input_dict['l1'])], True, True, False)]
    load = -input_dict['q1'] / 100
    beam_name = input_dict['name']
    b = Beam(beam_name, nodes, supports, section, load, discretization,
load_step, elements=elements_dict,
             data=input_dict)
    return b


def run_beam_list(beams, load_step):
    solved_beams = []
```

```
    for b in beams:
        b.add_elements()
        new_beam = solve_beam_incrementally(b, load_step)
        solved_beams.append(new_beam)
    return solved_beams


def run_from_app(input_dict):
    beam = create_beam_from_dict(input_dict, input_dict['discretization'],
input_dict['load_step'])
    beam = run_beam_list([beam, ], input_dict['load_step'])[-1]
    return beam


# App

class Application(tk.Tk):
    def __init__(self, *args, **kwargs):
        try:
            title = kwargs.pop('title')
        except KeyError:
            title = 'Untitled'
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, title)
        self.container = tk.Frame(self)
        self.container.grid(row=0, column=0)
        self.container.grid_rowconfigure(0, weight=1)
        self.container.grid_columnconfigure(0, weight=1)
        self.beam_dict = {}
        self.frames = {}
        self.active_frame = None
        for F in FRAMES:
            frame = F(self.container, self)
            self.frames[F.__name__] = frame
            frame.grid(row=0, column=0, sticky="nesw")
        self.menubar = Menu(self)
        self.config(menu=self.menubar)
        self.show_frame("StartPage")
        self.solved = False

    def show_frame(self, target_frame):
        self.active_frame = self.frames[target_frame]
        self.active_frame.tkraise()


class Menu(tk.Menu):
    def __init__(self, parent):
        tk.Menu.__init__(self, parent)
        self.add_command(label="Início", command=lambda:
parent.show_frame("StartPage"))
        self.add_command(label="Resultados", command=lambda:
parent.show_frame("OutputPage"))
        self.add_command(label="Sobre", command=lambda:
parent.show_frame("AboutPage"))


class BaseFrame(tk.Frame):
```

```
    instances = []

    def __init__(self, parent):
        tk.Frame.__init__(self, parent)
        BaseFrame.instances.append(self)


class AboutPage(BaseFrame):
    def __init__(self, parent, root):
        BaseFrame.__init__(self, parent)
        self.root = root
        row_counter = 0
        line_break = ttk.Label(self, text=' ')
        line_break.grid(row=row_counter, column=4)
        line_break.grid(row=row_counter, column=5)
        line_break.grid(row=row_counter, column=6)
        line_break.grid(row=row_counter, column=7)
        line_break.grid(row=row_counter, column=8)
        row_counter += 1
        label = ttk.Label(self, text="                          ",
font=LARGE_FONT)
        label.grid(row=row_counter, column=0)
        label = ttk.Label(self, text="Aviso:", font=LARGE_FONT)
        label.grid(row=row_counter, column=2)
        row_counter += 1

        heading1 = ttk.Label(self, text="Este aplicativo foi desenvolvido
como parte de um trabalho de conclusão")
        heading1.grid(row=2, column=2)
        heading2 = ttk.Label(self, text="de curso de engenharia civil pela
Universidade Federal de São Carlos.")
        heading2.grid(row=3, column=2)
        heading3 = ttk.Label(self, text="Trata-se de uma calculadora de
flechas imediatas, para vigas biapoiadas de")
        heading3.grid(row=4, column=2)
        heading4 = ttk.Label(self, text="concreto armado. O programa não
verifica se as solicitações na viga ")
        heading4.grid(row=5, column=2)
        heading5 = ttk.Label(self, text="ultrapassam a resistência máxima do
elemento estrutural. ")
        heading5.grid(row=6, column=2)
        heading5 = ttk.Label(self, text="Caso necessário, entre em contato
via email, pedrodeo@estudante.ufscar.br. ")
        heading5.grid(row=7, column=2)
        ttk.Label(self, text=' ').grid(row=8, column=3)

        self.p1 = ttk.Label(self, text="Lista de símbolos", font=LARGE_FONT)
        self.p1.grid(row=9, column=2)
        ttk.Label(self, text="Fck - Resistência à compressão característica
do concreto").grid(row=10, column=2)
        ttk.Label(self, text="As - Armadura positiva da viga").grid(row=11,
column=2)
        ttk.Label(self, text="As' - Armadura negativa da viga").grid(row=12,
column=2)
        ttk.Label(self, text="Fct - Resistência a tração do
concreto").grid(row=13, column=2)
        ttk.Label(self, text="Ic - Momento de inércia da seção de
```

```
concreto").grid(row=14, column=2)
        ttk.Label(self, text="Eci - Módulo de elasticidade do
concreto").grid(row=15, column=2)
        ttk.Label(self, text="Ecs - Módulo de elasticidade secante do
concreto").grid(row=17, column=2)
        ttk.Label(self, text="v - Coeficiente de Poisson").grid(row=18,
column=2)
        ttk.Label(self, text="alfa - Coeficiente de dilatação térmica do
concreto").grid(row=19, column=2)
        ttk.Label(self, text="b - Base da seção retangular").grid(row=20,
column=2)
        ttk.Label(self, text="h - Altura da seção retangular").grid(row=21,
column=2)
        ttk.Label(self, text="Taxa arm. - Taxa de armadura da
seção").grid(row=22, column=2)
        ttk.Label(self, text="d - Altura útil da seção").grid(row=23,
column=2)
        ttk.Label(self, text="Mr - Momento de fissuração").grid(row=24,
column=2)
        ttk.Label(self, text="alpha_e - Relação entre os módulos de
elasticidade do concreto e do aço").grid(row=25,

column=2)
        ttk.Label(self, text="x1 - profundidade da linha neutra no estádio
II").grid(row=26, column=2)
        ttk.Label(self, text="x2 - profundidade da linha neutra no estádio
II").grid(row=27, column=2)
        ttk.Label(self, text="x2/d - proporção entre profundidade da L.N. e a
altura útil da peça").grid(row=28,

column=2)
        ttk.Label(self, text="I1 - Inércia da seção homogeneizada no estádio
I").grid(row=29, column=2)
        ttk.Label(self, text="I2 - Inércia da seção homogeneizada no estádio
II").grid(row=30, column=2)


class OutputPage(BaseFrame):
    def __init__(self, parent, root):
        BaseFrame.__init__(self, parent)
        self.root = root
        self.beam = None
        self.heading = ttk.Label(self, text="Resultados da análise")
        self.heading.grid(row=1, column=2)
        beam_option_label = ttk.Label(self, text="Viga")
        beam_option_label.grid(row=2, column=1)
        self.beam_var = tk.StringVar(self)
        self.beam_var.set("")  # default value
        self.beam_var_option = tk.OptionMenu(self, self.beam_var, "")
        self.beam_var_option.grid(row=2, column=2)
        button1 = ttk.Button(self, text="Momento/Deformação", command=lambda:
self.plot_graph())
        button1.grid(row=4, column=1)
        button2 = ttk.Button(self, text="Momento Fletor", command=lambda:
self.show_bending_moment())
        button2.grid(row=4, column=2)
        button3 = ttk.Button(self, text="Força Cortante", command=lambda:
```

```
self.show_shear_forces())
        button3.grid(row=4, column=3)
        self.figure = Figure(figsize=(6, 5), dpi=100)
        self.figure.set_facecolor('grey')
        self.deflection = self.figure.add_subplot(111)
        self.canvas = FigureCanvasTkAgg(self.figure, self)
        self.canvas.draw()
        self.canvas.get_tk_widget().grid(row=5, column=1, columnspan=3)
        self.beam_description = ttk.Label(self, text='')
        self.beam_description.grid(row=5, column=4, )
        self.footing = ttk.Label(self, text="")
        self.footing.grid(row=6, column=2)
        self.save_msg = ttk.Label(self,
                                  text="Ao salvar uma viga com nome já
existente, os arquivos serão sobrescritos!")
        self.save_msg.grid(row=7, column=1, columnspan=3)
        save_button = ttk.Button(self, text="Salvar dados", command=lambda:
self.save_beam_data())
        save_button.grid(row=7, column=4)
        self.last_footer = ttk.Label(self, text="")
        self.last_footer.grid(row=8, column=2)

    def save_beam_data(self):
        if self.beam:
            if not os.path.isdir("vigas/"):
                os.mkdir("vigas/")
            if os.path.isdir("vigas/" + self.beam.name):
                rmtree("vigas/" + self.beam.name)
            os.mkdir("vigas/" + self.beam.name)
            os.mkdir("vigas/" + self.beam.name + "/etapas/")
            self.beam.save_analysis_data()
        self.save_msg.configure(text="Arquivos salvos em:
.../vigas/{}/".format(self.beam.name))

    def show_shear_forces(self):
        if self.beam:
            self.figure.clear()
            df =
self.beam.analysis.get_shear_diagram_dataframe(self.beam.total_length)
            self.beam_description.configure(text="Cortante Máxima: {:.2f}
kN".format(abs(min(df['shear']))))
            self.deflection = self.figure.add_subplot(111)
            self.deflection.plot(df['length'], df['shear'], color='b')
            self.deflection.plot(df['length'], [0 for _ in df['length']],
color='r')
            contour1 = [[0, 0], [0, df.at[0, 'shear']]]
            contour2 = [[self.beam.total_length, self.beam.total_length], [0,
-df.at[0, 'shear']]]
            self.deflection.plot(contour1[0], contour1[1], 'b')
            self.deflection.plot(contour2[0], contour2[1], 'b')
            self.canvas.draw()
            self.canvas.get_tk_widget().grid(row=5, column=1, columnspan=3)

    def show_bending_moment(self):
        if self.beam:
            self.figure.clear()
            df =
```

```
self.beam.analysis.get_bending_diagram_dataframe(self.beam.total_length)
            self.beam_description.configure(text="Mmáx: {:.2f}
kN.cm".format(abs(min(df['moment']))))
            self.deflection = self.figure.add_subplot(111)
            self.deflection.plot(df['length'], df['moment'], color='b')
            self.deflection.plot(df['length'], [0 for _ in df['length']],
color='r')
            self.canvas.draw()
            self.canvas.get_tk_widget().grid(row=5, column=1, columnspan=3)

    def refresh_options(self, time_now):
        self.last_footer.configure(text="Tempo de execução: {:.2f}
segundos.".format(time_now))
        if self.beam is None:
            return
        choice = self.beam.name
        self.beam_var_option['menu'].add_command(label=choice,
command=tk._setit(self.beam_var, choice))
        self.beam_var.set(choice)

    def plot_graph(self):
        if self.beam:
            self.figure.clear()
            self.beam_description.configure(text=self.beam.input_describe())
            df = self.beam.analysis.get_graph_dataframe()
            self.deflection = self.figure.add_subplot(111)
            self.deflection.plot(df['branson'], df['moment'], color='r')
            self.deflection.plot(df['bischoff'], df['moment'], color='g')
            self.deflection.plot(df['mef'], df['moment'], color='b')
            self.deflection.set_xlabel('Flecha')
            self.deflection.set_ylabel('Momento atuante')
            max_moment = max(abs(df['moment']))
            max_deflection = max(max(abs(df['bischoff'])),
max(abs(df['mef'])))
            grid_lines = [[[0, 0], [0, max_moment + 1000]], ]
            d = 0
            while d < max_deflection:
                d += 0.25
                grid_lines.append([[d, d], [0, max_moment + 1000]])
            d += 0.25
            m = 0
            while m < max_moment:
                grid_lines.append([[0, d], [m, m]])
                m += 1000
            grid_lines.append([[0, d], [m, m]])
            for g in grid_lines:
                self.deflection.plot(g[0], g[1], color='grey',
linewidth=0.25)
            legend_lines = [Line2D([0], [0], color='r', lw=4),
                            Line2D([0], [0], color='g', lw=4),
                            Line2D([0], [0], color='b', lw=4), ]
            self.deflection.legend(legend_lines, ['Branson', 'Bischoff',
'MEF-Branson'], loc='lower right')
            self.canvas.draw()
            self.canvas.get_tk_widget().grid(row=5, column=1, columnspan=3)
            self.footing.configure(text='Flecha imediata máxima calculada por
cada método:\n' +
```

```
                                        'Branson: {:.4f}
cm\n'.format(max(abs(df['branson']))) +
                                        'Bischoff: {:.4f}
cm\n'.format(max(abs(df['bischoff']))) +
                                        'MEF-Branson: {:.4f}
cm\n'.format(max(abs(df['mef'])))


class StartPage(BaseFrame):
    def __init__(self, parent, root):
        BaseFrame.__init__(self, parent)
        self.root = root
        row_counter = 0
        line_break = ttk.Label(self, text=' ')
        line_break.grid(row=row_counter, column=4)
        line_break.grid(row=row_counter, column=5)
        line_break.grid(row=row_counter, column=6)
        line_break.grid(row=row_counter, column=7)
        line_break.grid(row=row_counter, column=8)
        row_counter += 1
        label = ttk.Label(self, text="                          ",
font=LARGE_FONT)
        label.grid(row=row_counter, column=0)
        label = ttk.Label(self, text="Entre os dados abaixo",
font=LARGE_FONT)
        label.grid(row=row_counter, column=2)
        row_counter += 1
        line_break = ttk.Label(self, text=' ')
        line_break.grid(row=row_counter, column=1)
        row_counter += 1
        name_label = ttk.Label(self, text="Nome")
        name_label.grid(row=row_counter, column=2)
        self.beam_name_field = ttk.Entry(self, textvariable="name")
        self.beam_name_field.insert(0, "V001")
        self.beam_name_field.grid(row=row_counter, column=3)
        row_counter += 1
        line_break = ttk.Label(self, text=' ')
        line_break.grid(row=row_counter, column=1)
        row_counter += 1
        ttk.Label(self, text="Material").grid(row=row_counter, column=1)
        ttk.Label(self, text="Fck").grid(row=row_counter, column=2)
        self.fck_var = tk.StringVar(self)
        self.fck_var.set("50")
        fck_widget = tk.OptionMenu(self, self.fck_var, "20", "30", "35",
"40", "45", "50")
        fck_widget.grid(row=row_counter, column=3)
        ttk.Label(self, text="MPa").grid(row=row_counter, column=4)
        row_counter += 1
        line_break = ttk.Label(self, text=' ')
        line_break.grid(row=row_counter, column=1)
        row_counter += 1
        section_label = ttk.Label(self, text="Seção")
        section_label.grid(row=row_counter, column=1)
        section_x_label = ttk.Label(self, text="Base")
        section_x_label.grid(row=row_counter, column=2)
        self.section_x_field = ttk.Entry(self, textvariable="b")
        self.section_x_field.insert(0, "15")
```

```
        self.section_x_field.grid(row=row_counter, column=3)
        ttk.Label(self, text="cm").grid(row=row_counter, column=4)
        row_counter += 1
        section_y_label = ttk.Label(self, text="Altura")
        section_y_label.grid(row=row_counter, column=2)
        self.section_y_field = ttk.Entry(self, textvariable="h")
        self.section_y_field.insert(0, "35")
        self.section_y_field.grid(row=row_counter, column=3)
        ttk.Label(self, text="cm").grid(row=row_counter, column=4)
        row_counter += 1
        line_break = ttk.Label(self, text=' ')
        line_break.grid(row=row_counter, column=1)
        row_counter += 1
        rebar_label = ttk.Label(self, text='Armadura')
        rebar_label.grid(row=row_counter, column=1)
        rebar_posarea_label = ttk.Label(self, text='As')
        rebar_posarea_label.grid(row=row_counter, column=2)
        self.rebar_posarea_field = ttk.Entry(self, textvariable="as")
        self.rebar_posarea_field.insert(0, "1.51")
        self.rebar_posarea_field.grid(row=row_counter, column=3)
        ttk.Label(self, text="cm²").grid(row=row_counter, column=4)
        row_counter += 1
        rebar_negarea_label = ttk.Label(self, text="As'")
        rebar_negarea_label.grid(row=row_counter, column=2)
        self.rebar_negarea_field = ttk.Entry(self, textvariable="as1")
        self.rebar_negarea_field.insert(0, "1.51")
        self.rebar_negarea_field.grid(row=row_counter, column=3)
        ttk.Label(self, text="cm²").grid(row=row_counter, column=4)
        row_counter += 1
        rebar_cover_label = ttk.Label(self, text="Cobrimento")
        rebar_cover_label.grid(row=row_counter, column=2)
        self.rebar_cover_field = ttk.Entry(self, textvariable="cover")
        self.rebar_cover_field.insert(0, "5")
        self.rebar_cover_field.grid(row=row_counter, column=3)
        ttk.Label(self, text="cm").grid(row=row_counter, column=4)
        row_counter += 1
        line_break = ttk.Label(self, text=' ')
        line_break.grid(row=row_counter, column=1)
        row_counter += 1
        geometry_label = ttk.Label(self, text='Geometria')
        geometry_label.grid(row=row_counter, column=1)
        gap_label = ttk.Label(self, text='Vão 1')
        gap_label.grid(row=row_counter, column=2)
        self.gap_field = ttk.Entry(self, textvariable="gap1")
        self.gap_field.insert(0, "500")
        self.gap_field.grid(row=row_counter, column=3)
        ttk.Label(self, text="cm").grid(row=row_counter, column=4)
        row_counter += 1
        bar_count_label = ttk.Label(self, text='Número de barras')
        bar_count_label.grid(row=row_counter, column=2)
        self.bar_count_var = tk.StringVar(self)
        self.bar_count_var.set(20)
        bar_count_widget = tk.OptionMenu(self, self.bar_count_var, 4, 10, 20,
50, 100, 200)
        bar_count_widget.grid(row=row_counter, column=3)
        ttk.Label(self, text="n° barras").grid(row=row_counter, column=4)
        row_counter += 1
```

```
        line_break = ttk.Label(self, text=' ')
        line_break.grid(row=row_counter, column=1)
        row_counter += 1
        loads_label = ttk.Label(self, text='Carregamento')
        loads_label.grid(row=row_counter, column=1)
        load_type_label = ttk.Label(self, text='Tipo de Carregamento')
        load_type_label.grid(row=row_counter, column=2)
        self.load_type_var = tk.StringVar(self)
        self.load_type_var.set("Distribuído")
        load_type_widget = tk.OptionMenu(self, self.load_type_var,
"Distribuído")
        load_type_widget.grid(row=row_counter, column=3)
        row_counter += 1
        load1_label = ttk.Label(self, text='Carga')
        load1_label.grid(row=row_counter, column=2)
        self.load1_field = ttk.Entry(self, textvariable="qload1")
        self.load1_field.insert(0, "14")
        self.load1_field.grid(row=row_counter, column=3)
        ttk.Label(self, text="KN/m").grid(row=row_counter, column=4)
        row_counter += 1
        ttk.Label(self, text="Etapas de carga").grid(row=row_counter,
column=2)
        self.steps_var = tk.StringVar(self)
        self.steps_var.set("25")
        steps_widget = tk.OptionMenu(self, self.steps_var, "5", "10", "25",
"50", "100")
        steps_widget.grid(row=row_counter, column=3)
        row_counter += 1
        line_break = ttk.Label(self, text=' ')
        line_break.grid(row=row_counter, column=1)
        row_counter += 1
        button = ttk.Button(self, text="Calcular", command=lambda:
self.solve_beam())
        button.grid(row=row_counter, column=2)

    def get_field_values(self):
        disc = float(self.gap_field.get()) / int(self.bar_count_var.get())
        step = float(self.load1_field.get()) / int(self.steps_var.get())
        values = {'name': self.beam_name_field.get(),
                  'fck': int(self.fck_var.get()),
                  'yc': 1.4,
                  'gamma': 2500,
                  'v': 0.2,
                  'alpha': 1e-5,
                  'b': int(self.section_x_field.get()),
                  'h': int(self.section_y_field.get()),
                  'as': float(self.rebar_posarea_field.get()),
                  "asl": float(self.rebar_negarea_field.get()),
                  'cover': float(self.rebar_cover_field.get()),
                  'l1': int(self.gap_field.get()),
                  'l2': 0,
                  'q1': int(self.load1_field.get()),
                  'load_step': float(step),
                  'discretization': int(disc),
                  }
        return values
```

```
    def solve_beam(self):
        self.root.show_frame("OutputPage")
        start = time.time()
        beam = run_from_app(self.get_field_values())
        elapsed_time = time.time() - start
        self.root.solved = True
        self.root.frames["OutputPage"].beam = beam
        self.root.frames["OutputPage"].refresh_options(elapsed_time)
        self.root.frames["OutputPage"].plot_graph()


FRAMES = [StartPage, OutputPage, AboutPage]

if __name__ == "__main__":
    app = Application(title='ImediataVigas')
    app.geometry(RESOLUTION)
    app.mainloop()
```