UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA CIVIL DE INFRAESTRUTURA

LEONARDO RUDOLFO CARDOSO

INTRODUÇÃO À ANÁLISE ESTRUTURAL VIA MÉTODO DOS ELEMENTOS FINITOS

Joinville
2021

LEONARDO RUDOLFO CARDOSO

INTRODUÇÃO À ANÁLISE ESTRUTURAL VIA MÉTODO DOS ELEMENTOS FINITOS

Trabalho apresentado como requisito para obtenção do título de bacharel em Engenharia Civil de Infraestrutura do Centro Tecnológico de Joinville da Universidade Federal de Santa Catarina.

Orientadora: Dra. Anelize Borges Monteiro.

Joinville
2021

LEONARDO RUDOLFO CARDOSO

INTRODUÇÃO À ANÁLISE ESTRUTURAL VIA MÉTODO DOS ELEMENTOS FINITOS

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de bacharel em Engenharia Civil de Infraestrutura, na Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Joinville (SC), 07 de maio de 2021.

Banca Examinadora:

_____

Dra. Anelize Borges Monteiro
Orientadora/Presidente
Universidade Federal de Santa Catarina

_____

Dr. Diego Alexandre Duarte
Membro
Universidade Federal de Santa Catarina

_____

Dra. Débora Cordeiro Coelho Pinheiro
Membra
Pontifícia Universidade Católica de Minas Gerais

Dedico este trabalho a todos que, de alguma forma, fazem parte da minha vida.

Quem nunca errou nunca experimentou nada novo. (Albert Einstein).

RESUMO

Atualmente, a tecnologia permite a resolução de diversos problemas de engenharia por meio de métodos computacionais. Dentre eles, para a solução de equações diferenciais, o MEF é muito utilizado. De fato, existem diversos *softwares* que utilizam o procedimento numérico. No entanto, é fundamental o conhecimento básico do método implementado nesses programas, pois, só assim garante-se a correta aplicação pelo usuário. Além disso, tais aplicações requerem certa expertise matemática para serem desenvolvidas. Portanto, o estudo do MEF é imprenscindível para um engenheiro de estruturas. Tendo em vista essa necessidade, o presente trabalho visa levantar aspectos importantes e introduzir os conceitos iniciais para o estudo do MEF. Nesse sentido, o trabalho contempla a implementação computacional da formulação paramétrica do MEF e de elemento paramétrico bidimensional triangular de três nós. As implementações foram realizadas em Python e as malhas de elementos finitos foram geradas no programa GMesh. Apresenta-se um estudo da formulação paramétrica de MEF, a sua correlação com a metodologia de programação e a formulação dos modelos de análise implementados: estado plano de tensão e estado plano de deformação. O funcionamento adequado dos vários recursos implementados é comprovado por meio de dois exemplos numéricos nos quais é utilizado material elástico linear isotrópico. Com base no levantamento bibliográfico reunido, o algoritmo objetiva auxiliar outros acadêmicos a compreender o MEF de maneira introdutória, aliando a formulação teórica às implementações computacionais.

Palavras-chave: MEF. Problemas lineares-elásticos. Elementos triangulares de três nós.

ABSTRACT

Currently, the technology allows to solve engineering problems by computational methods. Among them, FEM is widely used in order to solve differential equations. In fact, there are many softwares that implement the numerical procedure. However, the basic knowledge about these methods is important to ensure its correct applications. Beyond that, these applications request a certain mathematical expertise to be developed. Therefore, studying FEM is essential to an structural engineer. In view of that need, this work aims to list importants aspects and introduce the initial concepts to FEM studying. In this way, the work contains a bidimensional parametric FEM computational implementation with three nodes triangles. The code is written in Python and the meshes are generated by Gmsh program. It is presented a FEM parametric formulation study, its correlation to programming methodology and the equations for the plane stress and plane strain analysis models. The algorithm is validated by two implementations of isotropic elastic material. Based on the bibliography presented, the algorithm presented aims assist other academics to understand the FEM in an introdutory way, allying the theorical formulation to the computational implementation.

Keywords: FEM. Linear elastic problems. Three node triangular elements.

LISTA DE FIGURAS

LISTA DE SIGLAS

CAD *Computer Aided Design*

EPD Estado Plano de Deformações

EPT Estado Plano de Tensões

MEF Método dos Elementos Finitos

PTV Princípio dos Trabalhos Virtuais

SUMÁRIO

1 INTRODUÇÃO

A análise estrutural consiste em resolver um problema de meio contínuo da estrutura real, substituindo-o por um modelo matemático por meio de hipóteses simplificadoras. Contudo, as equações diferenciais que regem estes modelos só possuem solução analítica em poucos casos simples. Mediante a dificuldade existente em solucioná-las, adotam-se modelos numéricos aproximados, que são chamados de discretos. Dentre os mais conhecidos, destaca-se o método de elementos finitos (MEF) (GONÇALVES, 2004).

O MEF, então, fornece uma forma de gerar algoritmos discretos para aproximação das soluções de equações diferenciais, a fim de serem calculados por um computador. O método requer, entretanto, uma certa habilidade matemática para implementação (BRENNER; SCOTT, 2008). Logo, o pleno conhecimento matemático do tema é indispensável para a criação de aplicações que utilizam método de elementos finitos.

Existem, atualmente, diversos programas comerciais que fazem o uso do método. De fato, conforme o avanço tecnológico presenciado nos últimos anos, a aplicação do MEF se tornou fundamental para estudos de engenharia em alto nível, proporcionando rápidas soluções de estruturas seguras e inovadoras. Contudo, tais programas estão condicionados a aplicações e interpretações de um usuário, havendo a possibilidade de equívocos decorrentes de um baixo nível de conhecimento sobre o MEF.

O presente trabalho, portanto, justifica-se pela importância da compreensão do MEF para o acadêmico de engenharia, que por vezes não têm a oportunidade de entrar em contato com o tema ao longo da graduação. Entender de que maneira os problemas são solucionáveis por meio do MEF é fundamental para garantir a interpretação adequada dos resultados obtidos por programas de dimensionamento estrutural, que poderão ser utilizados cotidianamente na vida profissional dos engenheiros e para fundamentar a implementação e o correto funcionamento de novos projetos computacionais.

Mediante a justificativa relatada, o desenvolvimento deste trabalho tem como intuito apresentar o MEF de maneira introdutória, aliando a formulação teórica às implementações computacionais. O levantamento bibliográfico reunido e os procedimentos referentes à implementação do MEF permitiram a elaboração de um algoritmo de solução para modelos de análise Estado Plano de Tensão (EPT) e Estado Plano de Deformação (EPD). Assim, poder-se-á contribuir para o estudo introdutório

sobre o tema de maneira prática e aplicada.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é aplicar o MEF para solucionar modelos de EPT e EPD de maneira prática e didática, relacionando os conceitos teóricos necessários para compreensão do método.

1.1.2 Objetivos Específicos

O objetivo geral é orientado pelos seguintes objetivos específicos:

- reunir a formulação paramétrica do Método dos Elementos Finitos e do elemento paramétrico triangular de três nós;
- desenvolver um algoritmo utilizando elementos finitos para resolução de problemas com os modelos de análise implementados: estado plano de tensão e estado plano de deformação;
- validar as implentações desenvolvidas e o funcionamento dos recursos por meio de exemplos numéricos, comparando os resultados obtidos com as respostas analíticas encontradas na literatura;
- auxiliar acadêmicos a compreender o MEF de maneira introdutória, aliando a formulação teórica às implementações computacionais.

2 FUNDAMENTAÇÃO TEÓRICA

O equacionamento dos problemas de mecânica dos sólidos e mecânica estrutural é fundamental para a modelagem matemática e simulação numérica. De maneira geral, sólidos e estruturas são projetados com a função de absorver e suportar carregamentos. As cargas impostas produzem tensão na estrutura do material, que responde de forma a aliviar essas tensões por meio de deformações. Conhecer de que maneira o sólido efetua essa atenuação permite prever deslocamentos e tensões (LIU; QUEK, 2003).

Com o intuito de introduzir os conceitos relacionados a essa sistemática, o presente capítulo traz equacionamentos para o Estado Plano de Tensão (EPT) e Estado Plano de Deformação (EPD) para materiais elástico-lineares isotrópicos, discretização dos modelos matemáticos e solução por meio do MEF.

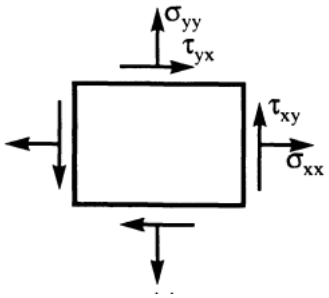
2.1 ESTADO PLANO DE TENSÃO E DEFORMAÇÃO

Problemas de sólidos tridimensionais podem, em certos casos, ser simplificados em modelos bidimensionais. Na verdade, é comum reduzir problemas de engenharia dessa forma. De maneira geral, assume-se que as variáveis do problema são independentes de um eixo, geralmente o z, perpendicular ao plano adotado (LIU; QUEK, 2003).

Existem duas situações de sólidos bidimensionais. Sólidos em estado plano de tensão possuem a espessura na direção z muito menor que as na direção x e y. Nesse caso, as forças externas atuam somente no plano xy. Logo, as tensões na direção z são nulas. Por outro lado, sólidos em estado plano de deformação são aqueles em que o comprimento na direção z é muito maior que os na direção x e y. Além disso, as forças externas são aplicadas uniformemente ao longo do eixo z, resultando em deformações no eixo z nulas (OÑATE, 2009).

Para introduzir o conceito de tensão e deformação, é comum a apresentação de um modelo de elemento infinitesimal, como ilustrado na Figura 1.

Figura 1 – Elemento infinitesimal 2D



Fonte: adaptado de Eschenauer et al. (1997, p. 19)

Considera-se que σ_{ij} é a tensão em que o índice i representa a superfície na qual a tensão atua, e o índice j a direção. Tomando como referência o sistema de coordenadas cartesianas, no caso bidimensional, há quatro componentes de tensão (visto que as componentes atreladas ao eixo z são dispensadas): as tensões normais σ_{xx} e σ_{yy} e as tensões de cisalhamento σ_{xy} e σ_{yx} (ou τ_{xy} e τ_{yx}). Como há a reciprocidade das componentes de cisalhamento no tensor de tensão, σ_{ij}=σ_{ji} (VECCI, 2017). Logo, $\sigma_{xy} = \sigma_{yx}$. O tensor de tensões pode ser descrito na forma matricial conforme Equação 1.

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{Bmatrix} \tag{1}$$

Quando submetido a uma tensão, o elemento infinitesimal sofrerá uma deformação como forma de absorver a energia transmitida, ou seja, para cada tensão, há uma deformação associada. As componentes de deformação são denotadas por ε_{ij}. Analogamente à tensão, o índice i diz respeito à face na qual a deformação ocorre e o índice j à direção da qual a deformação acontece. A deformação não é deslocamento, mas a mudança de forma no sólido, expressa na forma da grandeza adimensional ε.

Para um sistema de coordenadas cartesianas, no caso bidimensional, há quatro componentes de deformação (visto que as componentes atreladas ao eixo z são dispensadas): as deformações normais ε_{xx} e ε_{yy} e as deformações de cisalhamento ε_{xy}=ε_{yx}. A Equação 2 traz o tensor de deformações para o caso bidimensional.

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{Bmatrix} \tag{2}$$

Eschenauer et al. (1997) deduzem que, se u, e v são os deslocamentos nas coordenadas x e y, respectivamente, então as deformações são descritas na Equação 3.

$$\varepsilon_{xx} = \frac{\partial u}{\partial x}; \quad \varepsilon_{yy} = \frac{\partial v}{\partial y}; \quad \varepsilon_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{3}$$

Por fim, Liu e Quek (2003) escrevem a relação deformação-deslocamento na forma matricial, conforme Equação 4.

$$\boldsymbol{\varepsilon} = \boldsymbol{LU} \tag{4}$$

em que L é a matriz de operadores diferenciais (Equação 5) e U é a matriz de deslocamentos (Equação 6).

$$\boldsymbol{L} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \tag{5}$$

$$\boldsymbol{U} = \begin{bmatrix} u \\ v \end{bmatrix} \tag{6}$$

Para as análises estruturais, ainda são consideradas as equações constitutivas, definidas como expressões matemáticas que, por meio de propriedades dos materiais, relacionam tensão e deformação. Ou seja, elas dependem da constituição do material (LIU; QUEK, 2003). Se não ocorrem deformações permanentes quando aplicada uma tensão, diz-se que o regime é elástico. Além disso, se a taxa de deformação por tensão é constante, o material possui comportamento elástico linear (ESCHENAUER et al., 1997).

Karasudhi (1991) descreve o modelo elástico linear conforme Equação 7, em que $\boldsymbol{c}$ é a matriz constitutiva, composta por constantes inerentes do comportamento do material. Para a condição elástico linear em EPT, $\boldsymbol{c}$ é descrita pela Equação 8. Já para EPD, $\boldsymbol{c}$ é dada pela Equação 9.

$$\boldsymbol{\sigma} = \boldsymbol{c}\boldsymbol{\varepsilon} \tag{7}$$

$$\boldsymbol{c} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{8}$$

$$\boldsymbol{c} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{bmatrix} \tag{9}$$

em que E é o módulo de elasticidade do material e v o coeficiente de Poisson.

Uma vez determinada de que maneira o material se comporta diante de uma tensão, inicia-se a análise de equilíbrio estático do corpo. Seja $\boldsymbol{f}$ uma força que atua no elemento infinitesimal, segundo Eschenauer et al. (1997), as Equações 10 e 11 regem o equilíbrio estático para sólidos bidimensionais.

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + f_x = 0 \tag{10}$$

$$\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + f_y = 0 \tag{11}$$

Liu e Quek (2003) escrevem as Equações 10 e 11 na forma matricial como na Equação 12.

$$\boldsymbol{L^T}\boldsymbol{\sigma} + \boldsymbol{f_b} = \boldsymbol{0} \tag{12}$$

Substituindo a Equação 4 e 7 em 12, tem-se a Equação 13 (LIU; QUEK, 2003).

$$\boldsymbol{L^T c L U} + \boldsymbol{f_b} = \boldsymbol{0} \tag{13}$$

A resolução da Equação 13, segundo Liu e Quek (2003), é muito menos custosa computacionalmente quando comparado com o modelo tridimensional. Portanto,

quando possível, é muito vantajosa a simplificação de problemas estruturais em modelos bidimensionais.

2.2 MEF

Em problemas estruturais, é desejável determinar deslocamentos e tensões e a análise estrutural é o estudo que permite suprir tal necessidade. De maneira geral, o problema físico do meio contínuo da estrutura é substituído por um modelo matemático por meio de hipóteses simplificadoras. O resultado é um conjunto de equações diferenciais cuja solução analítica, nem sempre é possível. De fato, as soluções analíticas são conhecidas apenas para alguns poucos casos de equações diferenciais (MOREIRA, 2006).

A fim de evitar a complexidade relacionada à resolução analítica das equações diferenciais, faz-se a discretização do domínio e utiliza-se um modelo numérico aproximado. O MEF é um dos mais difundidos para tal (MOREIRA, 2006).

O MEF, então, é uma abordagem numérica para resolução de equações diferenciais que consiste na discretização do problema matemático. Dessa forma, é possível obter a solução aproximada da equação computacionalmente.

A equação do MEF para um problema estrutural bidimensional elástico pode ser escrita por meio do Princípio dos Trabalhos Virtuais (PTV) conforme Equação 14. (OÑATE, 2009)

$$\iint_A (\delta\varepsilon_x\sigma_x + \delta\varepsilon_y\sigma_y + \delta\varepsilon_{xy}\sigma_{xy})tdA = \iint_A (\delta u b_x + \delta v b_y)tdA + \oint_l (\delta u F_x + \delta v F_y)ds + \sum_i (\delta u_i P_{x_i} + \delta v_i P_{y_i}) \quad (14)$$

em que a integral ao lado esquerdo da equação representa o trabalho realizado pelas tensões σ_x, σ_y e σ_{xy} frente as deformações virtuais $\delta\varepsilon_x$, $\delta\varepsilon_y$, $\delta\varepsilon_{xy}$. Já a integral de área ao lado direito representa o trabalho virtual das forças de corpo b_x e b_y e a integral de linha referente a carregamentos na fronteira t_x e t_y. O somatório é a parcela de trabalho executado por meio de carregamentos externos pontuais P_{x_i} e P_{y_i}. Ainda, A representa a área e t a espessura do sólido 2D.

Reescrevendo 14 na forma matricial, segundo OÑate (2009), tem-se a Equação 15.

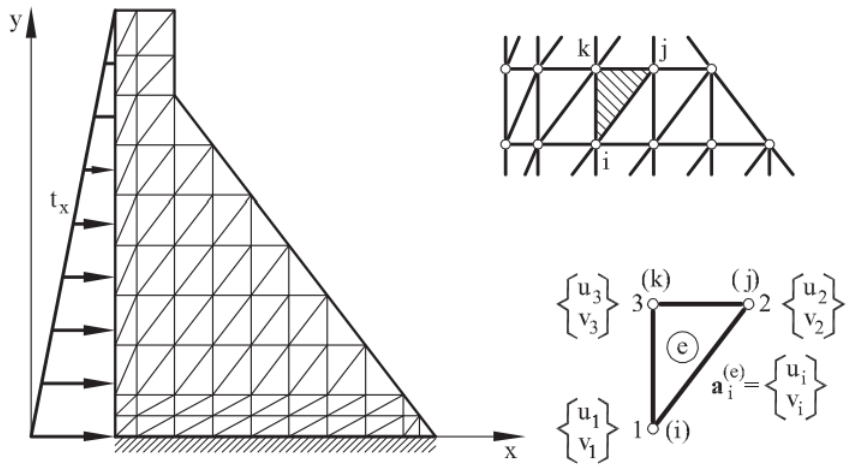
$$\iint_A \delta\boldsymbol{\varepsilon}^T\boldsymbol{\sigma}tdA - \iint_A \delta\boldsymbol{u}^T\boldsymbol{b}tdA - \oint_l \delta\boldsymbol{u}^T\boldsymbol{F}ds = \sum_i (\delta\boldsymbol{u}_i^T\boldsymbol{p}_i) \quad (15)$$

Parte-se então para a etapa de discretização do modelo, com o intuito de resolver a Equação 15 de forma aproximada.

2.3 DISCRETIZAÇÃO DO MODELO

Existem diversas formas de discretizar o modelo. A mais simples e introdutória é triangular linear. Divide-se, então, o modelo em diversos elementos finitos triangulares de três nós (ou três vértices). Cada nó possui 2 graus de liberdade, isto é, pode se deslocar em x e y (OÑATE, 2009). A Figura 2 ilustra a discretização com esse tipo de elemento finito.

Figura 2 – Discretização por elementos triangulares de 3 nós.



Fonte: adaptado de OÑate (2009).

Define-se que os deslocamentos nodais da estrutura global podem ser aproximados por uma função dos deslocamentos de cada elemento, chamada de função de forma. Ou seja

$$\boldsymbol{U}(x,y) \approx \boldsymbol{N}(x,y)\boldsymbol{u}^{\boldsymbol{e}} \tag{16}$$

em que $\boldsymbol{U}$ é o vetor coluna dos deslocamentos globais, $\boldsymbol{u}^{\boldsymbol{e}}$ dos deslocamentos locais do elemento e e $\boldsymbol{N}$ é uma matriz composta pelas chamadas funções de forma, descrita na Equação 17 (LIU; QUEK, 2003).

$$\boldsymbol{N} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix} \tag{17}$$

Para o elemento triangular linear é adotada uma função polinomial linear em x e em y. OÑate (2009) e Liu e Quek (2003) demonstram que as funções de forma podem ser escritas como na Equação 18.

$$N_i = a_i + b_i x + c_i y, \quad i = 1, 2, 3 \tag{18}$$

Dessa maneira, aproximam-se os deslocamentos globais como combinação linear dos deslocamentos nodais do elemento. Existem algumas formas de obter os coeficientes a_i, b_i e c_i. Liu e Quek (2003) demonstram que

$$\begin{aligned} a_i &= \frac{1}{2A_e}(x_j y_k - x_k y_j) \\ b_i &= \frac{1}{2A_e}(y_j - y_k) \\ c_i &= \frac{1}{2A_e}(x_k - x_j) \end{aligned} \tag{19}$$

em que A_e é a área do elemento triangular, dada pela Equação 20 e i, j e k variam em uma permutação cíclica de 1 a 3.

$$A_e = \frac{1}{2}[(x_2 y_3 - x_3 y_2) + (y_2 - y_3)x_1 + (x_3 - x_2)y_1] \tag{20}$$

Definido os deslocamentos, parte-se para o cálculo de suas derivadas para obtenção da matriz de deformação. Aplica-se, então, a Equação 16 em 4 (OÑATE, 2009). Tem-se, então

$$\boldsymbol{\varepsilon} = \boldsymbol{LNu} \tag{21}$$

em que $\boldsymbol{LN} = \boldsymbol{B}$. Fazendo a multiplicação matricial de $\boldsymbol{L}$ e $\boldsymbol{N}$ e aplicando as derivadas nas funções de forma, tem-se que

$$\boldsymbol{B} = \begin{bmatrix} b_1 & 0 & b_2 & 0 & b_3 & 0 \\ 0 & c_1 & 0 & c_2 & 0 & c_3 \\ c_1 & b_1 & c_2 & b_2 & c_3 & b_3 \end{bmatrix} \tag{22}$$

Então, utilizando a Equação 15 (PTV), para o elemento triangular tem-se que

$$\iint_A \delta\boldsymbol{\varepsilon}^T \boldsymbol{\sigma} t dA - \iint_A \delta\boldsymbol{u}^T \boldsymbol{b} t dA - \oint_l \delta\boldsymbol{u}^T \boldsymbol{F} ds = \delta\boldsymbol{u}_e^T \boldsymbol{q}_e \tag{23}$$

Como $\delta\boldsymbol{u}^T = \delta\boldsymbol{u_e}^T\boldsymbol{N}^T$ e $\delta\boldsymbol{\varepsilon}^T = \delta\boldsymbol{u_e}^T\boldsymbol{B}^T$, pode-se colocar os deslocamentos do elemento em evidência e arbitra-se os deslocamentos virtuais, ou seja,

$$\begin{gathered} \iint_A \delta\boldsymbol{u_e}^T \boldsymbol{B}^T \boldsymbol{\sigma} t dA - \iint_A \delta\boldsymbol{u_e}^T \boldsymbol{N}^T \boldsymbol{b} t dA - \oint_l \delta\boldsymbol{u_e}^T \boldsymbol{N}^T \boldsymbol{F} ds = \delta\boldsymbol{u}_e^T \boldsymbol{q}_e \\ \delta\boldsymbol{u_e}^T \left[\iint_A \boldsymbol{B}^T \boldsymbol{\sigma} t dA - \iint_A \boldsymbol{N}^T \boldsymbol{b} t dA - \oint_l \boldsymbol{N}^T \boldsymbol{F} ds \right] = \delta\boldsymbol{u}_e^T \boldsymbol{q}_e \\ \iint_A \boldsymbol{B}^T \boldsymbol{\sigma} t dA - \iint_A \boldsymbol{N}^T \boldsymbol{b} t dA - \oint_l \boldsymbol{N}^T \boldsymbol{F} ds = \boldsymbol{q}_e \end{gathered} \tag{24}$$

Por fim, OÑate (2009) substitui as tensões em função dos deslocamentos nodais

$$\iint_A \boldsymbol{B}^T \boldsymbol{c} \boldsymbol{B} \boldsymbol{u_e} t dA - \iint_A \boldsymbol{N}^T \boldsymbol{b} t dA - \oint_l \boldsymbol{N}^T \boldsymbol{F} ds = \boldsymbol{q}_e \tag{25}$$

Ainda, como as matrizes $\boldsymbol{B}$ e $\boldsymbol{c}$ e a espessura do elemento t é constante, tem-se que

$$\boldsymbol{K_e u_e} - \boldsymbol{f_e} = \boldsymbol{q_e} \tag{26}$$

em que $\boldsymbol{K_e} = \boldsymbol{B^T c B}$, chamada de matriz de rigidez do elemento e $\boldsymbol{f_e}$ é a soma das forças externas atuando no elemento (LIU; QUEK, 2003).

Definido a Equação de Equilíbrio 26, monta-se o Sistema Global de Equações.

2.4 SISTEMA GLOBAL

Segundo OÑate (2009), a equação de equilíbrio global pode ser deduzida considerando que os nós estão em equilíbrio, ou seja, a soma de todas as forças em cada nó j anulam os carregamentos pontuais p_j. Portanto, tem-se que

$$\sum_e \boldsymbol{q}_{e_i} = \boldsymbol{p_j}, \quad j = 1, N \tag{27}$$

em que $\boldsymbol{q}_{e_i}$ é o vetor de forças nodais atuantes no nó i do elemento e e $\boldsymbol{p_j}$ o vetor de forças nodais resultantes na estrutura global. A montagem da matriz de rigidez global é feita de maneira análoga.

Busca-se, então, solucionar o sistema linear global (Equação 28) calculando as contribuições dos elementos e montando a matriz de rigidez global da estrutura.

$$\boldsymbol{K u} = \boldsymbol{f} \tag{28}$$

em que $\boldsymbol{K}$ é a matriz de rigidez da estrutura, $\boldsymbol{u}$ é o vetor de deslocamentos nodais e $\boldsymbol{f}$ é o vetor de cargas externas. Para haver solução da Equação 28, é necessário a imposição das condições de contorno da estrutura. Estas podem ser impostas em forma essencial (deslocamentos nodais) ou em forma natural (forças nodais) (ALMEIDA, 2005). Impostas as condições de contorno, o sistema se torna particionado. Tem-se que

$$\begin{bmatrix} [\boldsymbol{K_{ff}}] & [\boldsymbol{K_{fc}}] \\ [\boldsymbol{K_{cf}}] & [\boldsymbol{K_{cc}}] \end{bmatrix} \begin{bmatrix} [\boldsymbol{U_f}] \\ [\boldsymbol{U_c}] \end{bmatrix} = \begin{bmatrix} [\boldsymbol{F_f}] \\ [\boldsymbol{F_c}] \end{bmatrix} \tag{29}$$

ou,

$$\begin{aligned} \boldsymbol{K_{ff}U_f} + \boldsymbol{K_{fc}U_c} &= \boldsymbol{F_f} \\ \boldsymbol{K_{cf}U_f} + \boldsymbol{K_{cc}U_c} &= \boldsymbol{F_c} \end{aligned} \tag{30}$$

em que os índices f (*free*) são os graus de liberdade livres da estrutura e c (*constrained*) os restritos. Primeiramente, calculam-se os deslocamentos desconhecidos $\boldsymbol{U_f}$, e depois as forças nodais restritas $\boldsymbol{F_c}$.

3 MÉTODOS

O trabalho inicia na pesquisa bibliográfica. Sua essência é permitir obtenção de dados (ZANELLA, 2013). Nessa etapa, os conceitos e equacionamentos acerca do método dos elementos finitos foram levantados utilizando livros e artigos publicados, fato que garante a integridade científica.

Uma vez mapeado o funcionamento do MEF, desenvolveu-se um algoritmo para aplicação do método de elementos finitos para um problema de estado plano de tensão elástico linear. Para desenvolvimento, Python foi escolhido como linguagem de programação devido ter sintaxe simples e intuitiva, além de diversas bibliotecas científicas de código aberto.

3.1 PROBLEMAS PARA VALIDAÇÃO DO ALGORITMO

Para a validação do código, foram escolhidos dois problemas elástico lineares com materiais isotrópicos. Um problema é de viga parede, em que se considera o modelo de análise de EPT e outro de uma barragem, em que se considera o modelo de análise de EPD. Ambos os exemplos foram escolhidos com base em Almeida (2005) e serão detalhados no Capítulo 4.

Para executar o algoritmo nas condições exigidas pelo modelo criado é necessário prover as informações sobre as hipóteses simplificadoras relacionadas ao material, carregamento, condições de contorno e geometria que gerem a malha e os atributos do modelo.

Na etapa de montagem e resolução do modelo, combinam-se as informação matematicamente representadas, de modo a produzir equações algébricas que, quando solucionadas, permitem obter as diversas grandezas. Na avalição de resultados, o analista faz crítica e verifica a adequação dos mesmos ao problema em estudo (ALMEIDA, 2005).

Neste trabalho, a etapa (1) de criação do modelo (pré-processamento) foi realizada utilizando Gmsh e Python. Gmsh é um gerador de malhas para elementos finitos com *CAD* (*Computer Aided Design*) e pós-processamento embutido (GEUZAINE; REMACLE, 2009). O *software* foi escolhido por ser de código aberto, possuir interface gráfica e fácil aplicação. O Item 3.2 descreve o pré-processamento.

3.2 PRÉ-PROCESSAMENTO COM GMSH E PYTHON

O pré-processamento realizado no Gmsh consistiu em desenhar as estruturas dos problemas tratados, aplicar os grupos físicos desejados para posterior identificação nas condições de contorno e gerar as malhas de elementos finitos conforme os algoritmos disponíveis. O programa é dividido em módulos, e para esta etapa foram utilizados o *geometry* para desenho e grupos físicos e o *meshing* para gerar a malha do modelo. As geometrias criadas são salvas em arquivos de extensão .geo, enquanto as malhas utilizam .msh.

Após gerar o arquivo .msh com a malha do modelo, é necessário obter as informações contidas no arquivo e aplicar no algoritmo de MEF. A documentação do Gmsh é ampla e contempla de que forma os arquivos .msh são escritos. Tendo em vista a necessidade da transição, foram desenvolvidos módulos em Python para realizar a tarefa. Uma vez feita a leitura dos arquivos, os dados estão prontos para serem inseridos no modelo de MEF implementado.

3.3 MEF EM PYTHON

Para a compreensão da etapa (2) de montagem e resolução do modelo, serão, serão apresentados alguns passos e trechos cruciais do código implementado. No decorrer do código, nota-se o uso da biblioteca *numpy* abreviada em *np*, amplamente utilizada e difundida no Python.

Primeiramente, faz-se as definições de *input* das propriedades do material, qual o modelo de análise (EPT ou EPD), qual a aceleração de corpo a qual o material é submetido, os caminhos para o arquivo .msh e onde se deseja salvar os resultados, como descrito na Figura 3.

Figura 3 – Propriedades do material e malha.

```
geometry_label = "beam"
analysis_type = 'plane_stress'

E = 2e5
v = 0.30
h = 0.1
d = 0

material_props = Props2D(E, v, h, d)

g = -9.81

# body acceleration
ba = np.array([
    0, g, 0, g, 0, g
])

file_name = "geometry.msh"
dirname = path.dirname(__file__)

mesh_path = path.join(
    dirname,
    "pre_processing",
    "meshing",
    geometry_label,
    file_name
)
result_path = path.join(
    dirname,
    "post_processing",
    geometry_label
)
```

Fonte: O autor (2021).

Posteriormente, define-se uma função para as condições de contorno. A imposição das condições de contorno é particular de cada problema. De maneira geral, faz-se a imposição de forças e deslocamentos prescritos. A função deve receber os nós do Gmsh e os vetores $\boldsymbol{U}$ e $\boldsymbol{F}$, aplicar as condições de contorno (prescrição de deslocamentos em $\boldsymbol{U}$ e aplicação de forças em $\boldsymbol{F}$) e retornar os vetores resultantes junto com os valores dos graus de liberdade restringidos. Para cada imposição de deslocamento feito, deve ser adicionado o grau de liberdade correspondente em "cdof". A Figura 4 mostra um exemplo implementado para o problema da viga parede, tratado detalhadamente no Capítulo 4.

Figura 4 – Função para aplicação das condições de contorno na viga parede.

```
def boundary_conditions(U, F, nodes):
    embeddedNodes = []
    fixYNodes = []
    loadedNodes = []

    for node in nodes:
        physicalGroups = node.entity.physicalGroups
        for physicalGroup in physicalGroups:
            if physicalGroup.name == '"Embedded"':
                embeddedNodes.append(node.tag)
            if physicalGroup.name == '"Loaded"':
                loadedNodes.append(node.tag)
            if physicalGroup.name == '"FixY"':
                fixYNodes.append(node.tag)

    cdof = []
    u0 = 0
    f0 = -1

    for nodetag in embeddedNodes:
        cdof.append(2*(nodetag-1))
        cdof.append(2*(nodetag-1)+1)
        U[2*(nodetag-1)] = u0
        U[2*(nodetag-1)+1] = u0

    for nodetag in fixYNodes:
        cdof.append(2*(nodetag-1)+1)
        U[2*(nodetag-1)+1] = u0

    for i, nodetag in enumerate(loadedNodes):
        if i==len(loadedNodes)-1:
            break
        node0ind = nodetag-1
        node1ind = nodetag
        node0 = nodes[node0ind]
        node1 = nodes[node1ind]
        l = np.abs(node0.coordinate.componentX - node1.coordinate.componentX)
        F[2*(node0ind)+1] += f0 * l / 2
        F[2*(node1ind)+1] += f0 * l / 2

    return U, F, cdof
```

Fonte: O autor (2021).

Definidos estes parâmetros, parte-se para a construção do objeto *Solver*. *Solver* é a classe implementada para executar a solução via MEF de problemas bidimensionais de materiais elástico lineares isotrópicos. O objeto criado contempla diversos métodos e atributos, divididos e criados com o intuito de facilitar a leitura e organização do código.

O método *__init__* da classe *Solver* (Figura 5) possui todas as inicializações das variáveis envolvidas no MEF. Nota-se que a implementação segue o conceito de encapsulamento na inicialização das variáveis com início "__", cujo objetivo é progeter as variáveis do objeto para garantir a integridade do modelo. Todas essas propriedades são definidas no escopo da classe, para que o implementador do objeto possa utilizá-las, mas não sobrescrevê-las.

Figura 5 – Inicialização de variáveis no objeto *Solver*.

```
def __init__(
    self,
    analysis_type,
    mesh_path,
    result_path,
    material_props,
    boundary_conditions,
    ba
    ):
    self.__analysis_type = analysis_type
    self.__mesh_path = mesh_path
    self.__result_path = result_path
    self.__material_props = material_props
    self.__boundary_conditions = boundary_conditions
    self.__ba = ba
    self.__gmshr = read_gmsh(mesh_path, False)
    self.__triangle_tags = [
        element.tag for element in self.gmshr.elements
        if element.elementType == 2
    ]
    self.__triangles = [
        Triangle(
            element.nodes[0].coordinate,
            element.nodes[1].coordinate,
            element.nodes[2].coordinate
        )
        for element in self.gmshr.elements if element.elementType == 2
    ]
    self.__n_dofs = 2*len(self.gmshr.nodes)
    self.__adof = np.arange(self.n_dofs)
    self.__K = np.zeros((self.n_dofs, self.n_dofs))
    self.__U = np.zeros((self.n_dofs))
    self.__F = np.zeros((self.n_dofs))
    self.__n_elements = len(self.gmshr.elements)
    self.__Sxx = np.zeros(self.n_elements)
    self.__Syy = np.zeros(self.n_elements)
    self.__Sxy = np.zeros(self.n_elements)

    self.__Exx = np.zeros(self.n_elements)
    self.__Eyy = np.zeros(self.n_elements)
    self.__Exy = np.zeros(self.n_elements)

    if (analysis_type == 'plane_stress'):
        self.__element_stiffness = lts.element_stiffness_plane_stress
        self.__strain_and_stress = lte.strain_and_stress_plane_stress
    elif (analysis_type == 'plane_strain'):
        self.__element_stiffness = lts.element_stiffness_plane_strain
        self.__strain_and_stress = lte.strain_and_stress_plane_strain
    else:
        raise ValueError("analysis_type {} not supported".format(analysis_type))
```

Fonte: O autor (2021).

A função read_gmsh faz a leitura do arquivo .msh e salva as informações em um objeto "GmshReader", que possui propriedades contendo as informações da malha. A partir dela, pode-se consumir os dados dos elementos e nós.

Para solucionar o problema basta chamar o método *solve* (Figura 6). Então, o algoritmo executa outros cinco métodos, que constituem as etapas: aplicação das condições de contorno, montagem da matriz de rigidez, particionamento do sistema global, solução do sistema linear e montagem das matrizes de deformação e tensão.

Figura 6 – Método *solve*.

```
def solve(self):
    self.apply_boundary_conditions()
    self.assemble_stiffness()
    self.partitionate_system()
    self.solve_system()
    self.assemble_strain_stress()
```

Fonte: O autor (2021).

O primeiro método *self.apply_boundary_conditions* consiste em executar a função de condições de contorno passada como argumento pelo implementador (Figura 7) e definir os graus de liberdade restritos e os livres.

Figura 7 – Aplicação das condições de contorno.

```
def apply_boundary_conditions(self):
    self.__U, self.__F, self.__cdof = self.__boundary_conditions(
        self.U,
        self.F,
        self.gmshr.nodes
    )
    self.__fdof = list(set(self.adof)-set(self.cdof))
```

Fonte: O autor (2021).

A etapa da montagem da matriz de rigidez é descrita na Figura 8. No *foor loop* é feito o cálculo da matriz de rigidez de cada elemento e já se faz a adição na matriz de rigidez global $\boldsymbol{K}$. Nota-se que, para a correspondência de cada nó do elemento com o nó global é utilizado um vetor $\boldsymbol{d}$. O vetor $\boldsymbol{d}$ contém as posições de cada grau de liberdade do elemento na matriz de rigidez global. A subtração -1 das tags dos nós é feita para corrigir a diferença entre o Python e Gmsh. No Python, a contagem padrão é realizada a partir do zero, enquanto no Gmsh é utilizado um.

Figura 8 – Montagem da matriz de rigidez.

```
def assemble_stiffness(self):
    for element in self.gmshr.elements:
        if element.elementType == 2:
            nodes = element.nodes

            xy = np.array([
                [
                    node.coordinate.componentX,
                    node.coordinate.componentY
                ]
                for node in nodes
            ])

            ke, fe = self.element_stiffness(xy, self.ba, self.material_props)

            d = [
                2*(nodes[0].tag-1),
                2*(nodes[0].tag-1)+1,
                2*(nodes[1].tag-1),
                2*(nodes[1].tag-1)+1,
                2*(nodes[2].tag-1),
                2*(nodes[2].tag-1)+1
            ]

            for j in range(6):
                p = d[j]
                self.__F[p] += fe[j]
                for k in range(6):
                    q = d[k]
                    self.__K[p, q] += ke[j, k]
```

Fonte: O autor (2021).

A matriz de rigidez de cada elemento k_e é montada por meio da função *element_stiffness_plane_stress* ou *element_stiffness_plane_strain* contidas na classe *LinearTriangularStiffness*, conforme Figura 9.

Figura 9 – Cálculo da matriz de rigidez do elemento triangular linear.

```
class LinearTriangularStiffness:
    @staticmethod
    def element_stiffness_plane_stress(
        xy,
        ba,
        props: Props2D
    ):

        E = props.youngModulus
        v = props.poisson
        h = props.thickness
        d = props.density

        Ae = ltsf.area(xy)
        B = ltsf.assemble_b_matrix(xy)

        ke = h*Ae*B.T.dot(lte.plane_stress_C(E, v)).dot(B)

        fe = d*h*(Ae/3.0)*ba # verificar

        return ke, fe

    @staticmethod
    def element_stiffness_plane_strain(
        xy,
        ba,
        props: Props2D
    ):

        E = props.youngModulus
        v = props.poisson
        h = props.thickness
        d = props.density

        Ae = ltsf.area(xy)
        B = ltsf.assemble_b_matrix(xy)

        ke = h*Ae*B.T.dot(lte.plane_strain_C(E, v)).dot(B)

        fe = d*h*(Ae/3.0)*ba # verificar

        return ke, fe
```

Fonte: O autor (2021).

Os valores de A_e (área do elemento triangular) e $\boldsymbol{B}$ (derivadas das funções de forma) são calculados na classe *LinearTriangularShapeFunctions*, conforme Figura 10. Já as matrizes constitutivas foram implementadas na classe *LinearTriangularElasticity* (Figura 11).

Figura 10 – Cálculo da área e da matriz B do elemento triangular linear.

```
class LinearTriangularShapeFunctions:

    @staticmethod
    def area(xy: list):
        x = xy[:, 0]
        y = xy[:, 1]

        P = np.array([
            [1, x[0], y[0]],
            [1, x[1], y[1]],
            [1, x[2], y[2]]
        ])

        Ae = 0.5*np.abs(np.linalg.det(P))

        return Ae

    @staticmethod
    def assemble_b_matrix(xy: list):
        x = xy[:, 0]
        y = xy[:, 1]

        Ae = LinearTriangularShapeFunctions.area(xy)

        b1 = y[1] - y[2]
        b2 = y[2] - y[0]
        b3 = y[0] - y[1]

        c1 = x[2] - x[1]
        c2 = x[0] - x[2]
        c3 = x[1] - x[0]

        B = (0.5/Ae)*np.array([
            [b1,    0, b2,    0, b3,    0],
            [   0, c1,    0, c2,    0, c3],
            [c1, b1, c2, b2, c3, b3]
        ])

        return B
```

Fonte: O autor (2021).

Figura 11 – Cálculo das matrizes constitutivas.

```
class LinearTriangularElasticity:
    @staticmethod
    def plane_stress_C(E, v):
        C = (E/(1-v**2)) * np.array([
            [1, v, 0],
            [v, 1, 0],
            [0, 0, (1-v)/2]
        ])
        return C

    @staticmethod
    def plane_strain_C(E, v):
        C = (E*(1-v))/(1+v)/(1-2*v) * np.array([
            [1, v/(1-v), 0],
            [v/(1-v), 1, 0],
            [0, 0, (1-2*v)/(2*(1-v))]
        ])
        return C
```

Fonte: O autor (2021).

Uma vez montada a matriz de rigidez global, vetores de deslocamentos e de força, pode-se fazer o particionamento do sistema. A Figura 12 mostra a maneira que esta etapa foi implementada utilizando *numpy*.

Figura 12 – Particionamento da matriz de rigidez.

```
def partitionate_system(self):
    self.__Kff = self.K[np.ix_(self.fdof, self.fdof)]
    self.__Kfc = self.K[np.ix_(self.fdof, self.cdof)]
    self.__Kcf = self.K[np.ix_(self.cdof, self.fdof)]
    self.__Kcc = self.K[np.ix_(self.cdof, self.cdof)]
    self.__Uf = self.U[self.fdof]
    self.__Uc = self.U[self.cdof]
    self.__Ff = self.F[self.fdof]
    self.__Fc = self.F[self.cdof]
```

Fonte: O autor (2021).

O sistema linear, finalmente, pode ser resolvido. Para isso, utilizou-se o método *solve* do *package scipy.linalg*. Encontra-se os valores de deslocamentos livres U_f e substitui-se na Equação 30 para obter as reações de apoio. A Figura 13 mostra essa etapa.

Figura 13 – Solução dos deslocamentos e reações de apoio.

```
def solve_system(self):
    self.__Uf = solve(self.Kff, self.Ff-self.Kfc.dot(self.Uc))
    self.__U[self.fdof] = self.Uf

    self.__Fc = self.Kcf.dot(self.Uf) + self.Kcc.dot(self.Uc)
    self.__F[self.cdof] = self.Fc
```

Fonte: O autor (2021).

Finalizada a solução do problema por elementos finitos, parte-se para o cálculo das tensões e deformações para serem apresentadas em gráficos no pós-processamento. Então, itera-se os elementos filtrando os triângulos para calcular as deformações e as tensões no elemento e já se adicionam as parcelas na matriz de global, similarmente ao processo de rigidez. A Figura 14 mostra a implementação.

Figura 14 – Montagem das matrizes de tensões e deformações.

```
def assemble_strain_stress(self):
    for element in self.gmshr.elements:
        if element.elementType == 2:
            nodes = element.nodes

            xy = np.array([
                [
                    node.coordinate.componentX,
                    node.coordinate.componentY
                ]
                for node in nodes
            ])

            d = [2*(nodes[0].tag-1),
                2*(nodes[0].tag-1)+1,
                2*(nodes[1].tag-1),
                2*(nodes[1].tag-1)+1,
                2*(nodes[2].tag-1),
                2*(nodes[2].tag-1)+1]

            ue = self.U[d]

            Ee, Se = self.strain_and_stress(
                xy, self.material_props.youngModulus,
                self.material_props.poisson,
                ue
            )

            self.__Sxx[element.tag-1] = Se[0]
            self.__Syy[element.tag-1] = Se[1]
            self.__Sxy[element.tag-1] = Se[2]
            self.__Exx[element.tag-1] = Ee[0]
            self.__Eyy[element.tag-1] = Ee[1]
            self.__Exy[element.tag-1] = Ee[2]
```

Fonte: O autor (2021).

Nota-se que as tensões e deformações também são calculadas na classe *LinearTriangularElasticity* (Figura 15).

Figura 15 – Montagem das matrizes de tensões e deformações.

```
class  LinearTriangularElasticity
    @staticmethod
    def strain_and_stress_plane_stress(xy: list, E, v, Ue):
        B = ltsf.assemble_b_matrix(xy)
        e = B.dot(Ue)
        s = LinearTriangularElasticity.plane_stress_C(E, v).dot(B).dot(Ue)

        return e, s

    @staticmethod
    def strain_and_stress_plane_strain(xy: list, E, v, Ue):
        B = ltsf.assemble_b_matrix(xy)
        e = B.dot(Ue)
        s = LinearTriangularElasticity.plane_strain_C(E, v).dot(B).dot(Ue)

    return e, s
```

Fonte: O autor (2021).

Com os valores de tensões e deformações calculados para cada elemento, inicia-se a etapa de pós-processamento.

3.4 PÓS-PROCESSAMENTO COM GMSH E PYTHON

Para utilizar o pós-processamento no Gmsh é necessário salvar os valores de tensões e deformações obtidos para cada elemento em um arquivo .msh. A documentação do Gmsh possui exemplos e explica de que forma o arquivo deve ser escrito. Para isso, foi desenvolvida uma função no Python que faz a escrita conforme a documentação. A Figura 16 traz a implementação.

Figura 16 – Função para escrita das tensões e deformações num arquivo .msh.

```
def write_element_result(
    file: str,
    label: str,
    result: list,
    tags: list
):
    fout = open(file, "w")

    fout.write("$MeshFormat\n")
    fout.write("4.1 0 8\n")
    fout.write("$EndMeshFormat\n")
    fout.write("$ElementData\n")
    fout.write("1\n")
    fout.write('"{}"\n'.format(label))
    fout.write("1\n")
    fout.write("0.0\n")
    fout.write("3\n")
    fout.write("0\n")
    fout.write("1\n")
    fout.write("{}\n".format(len(tags)))

    for tag in tags:
        fout.write("{} {}\n".format(tag, result[tag-1]))

    fout.write("$EndElementData\n")

    fout.close()
```

Fonte: O autor (2021).

No Gmsh, então, faz-se um *merge* do arquivo .msh que contém a malha do sólido com o .msh que contém os valores de tensão e deformação calculados para os elementos.

4 RESULTADOS

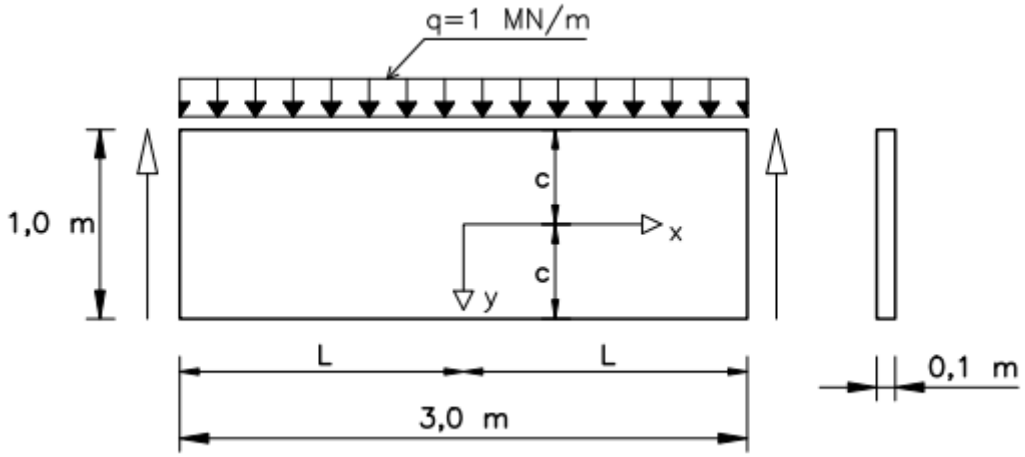
Neste capítulo são apresentados dois problemas bidimensionais discretizados com elementos finitos triangulares que visam verificar e validar os recursos implementados neste trabalho: a viga parede avaliada com o modelo de análise em EPT e a barragem analisada em EPD. As soluções obtidas pelo programa desenvolvido são comparadas com as soluções analíticas correspondentes e com os resultados numéricos obtidos por Almeida (2005).

4.1 EXEMPLOS DE VERIFICAÇÃO

4.1.1 Viga parede

O problema da viga parede é ilustrado na Figura 17. O objetivo da implementação da análise numérica é obter resultados semelhantes à solução exata deste problema, conforme encontrado em Timoshenko e Goodier (1951). Dessa forma, é possível validar o algoritmo implementado para EPT. O material da viga é elástico linear, com módulo de elasticidade $E = 2 \times 10^5$ MPa e coeficiente de Poisson $\nu = 0,3$. A condição de contorno essencial é fixação em x e y no nó com coordenada (-0.5, -1.5) e fixação em y nas extremidades longitudinais da viga.

Figura 17 – Problema de viga parede

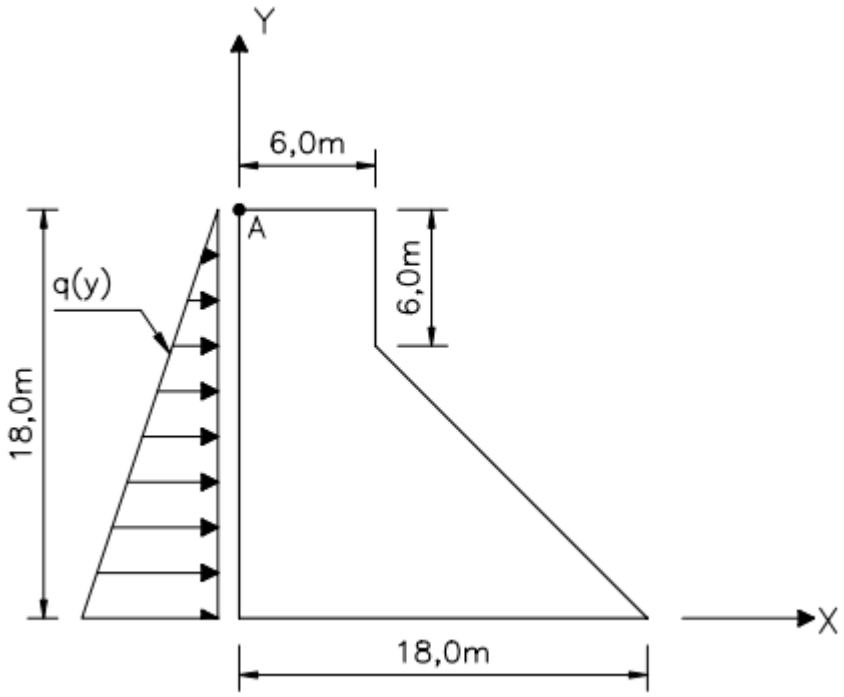


Fonte: Almeida (2005).

4.1.2 Barragem

O problema da barragem é ilustrado na Figura 18. O objetivo da implementação da análise numérica é obter resultados semelhantes aos apresentados por Almeida (2005) e validar o algoritmo implementado para EPD. O material é considerado elástico linear, com módulo de elasticidade $E = 20,8 \times 10^5 MPa$ e coeficiente de Poisson $\nu = 0,2$. O carregamento é hidrostático, conforme função $q(y) = 180 - 10y$ (*kN/m/m*). A condição de contorno essencial é a fixação em x e y dos nós da base da barragem.

Figura 18 – Problema de viga parede

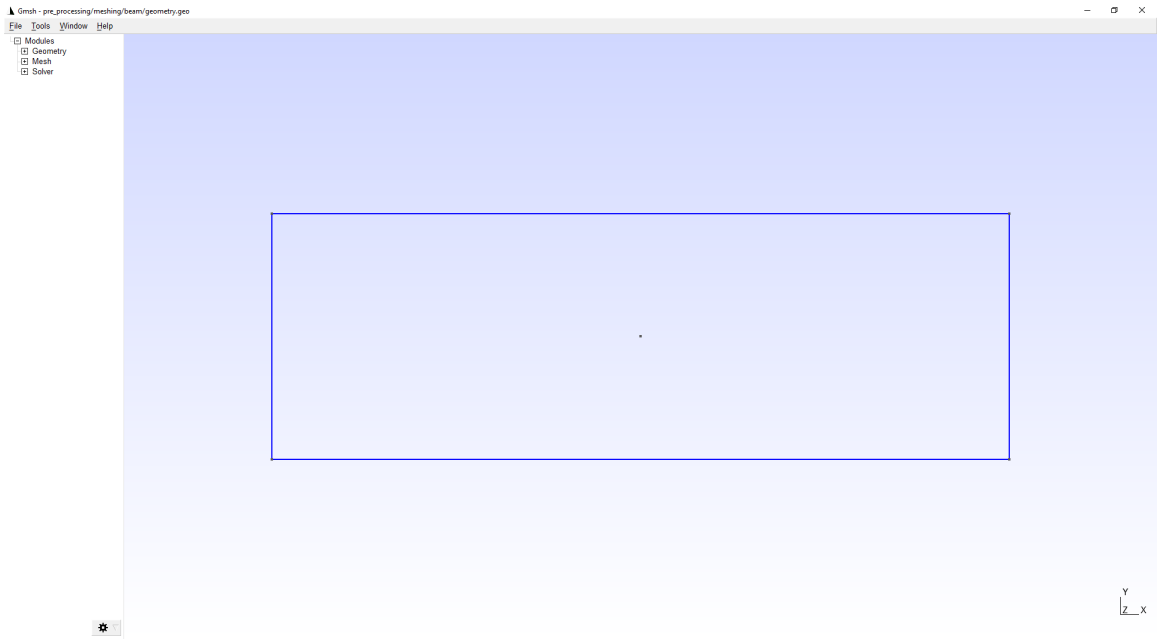


Fonte: Almeida (2005).

4.2 PRÉ-PROCESSAMENTO COM GMSH E PYTHON

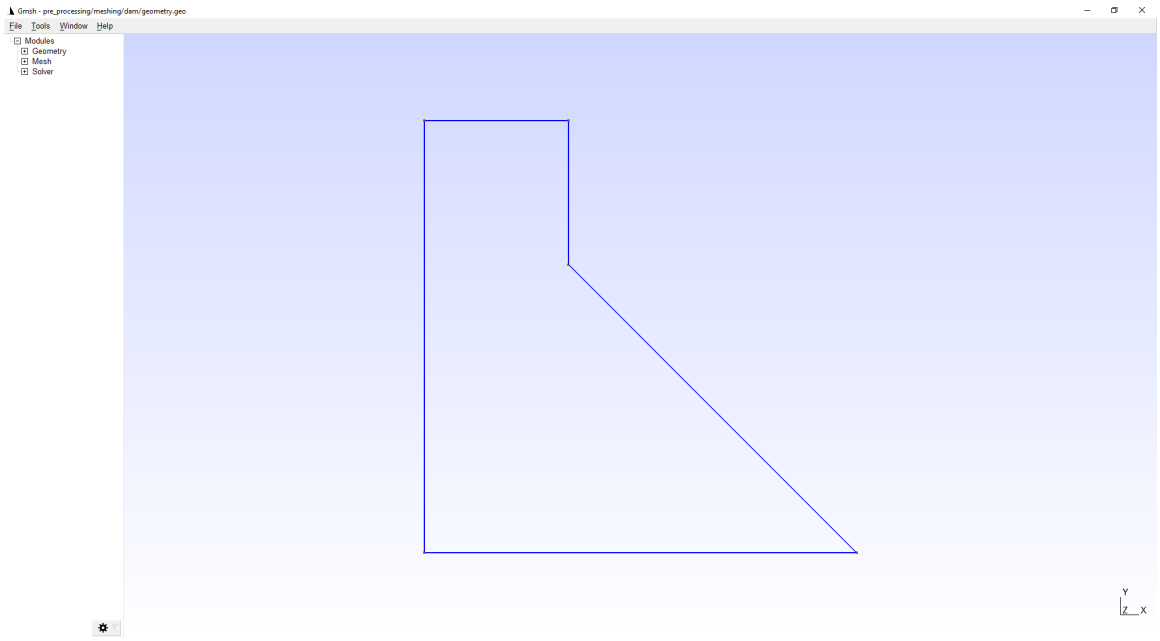
O pré-processamento com o Gmsh foi executado com o CAD integrado. As Figuras 19 e 20 mostram as geometrias desenhadas para o problema da viga parede e da barragem, respectivamente.

Figura 19 – Geometria criada para o problema da viga parede.



Fonte: O autor (2021).

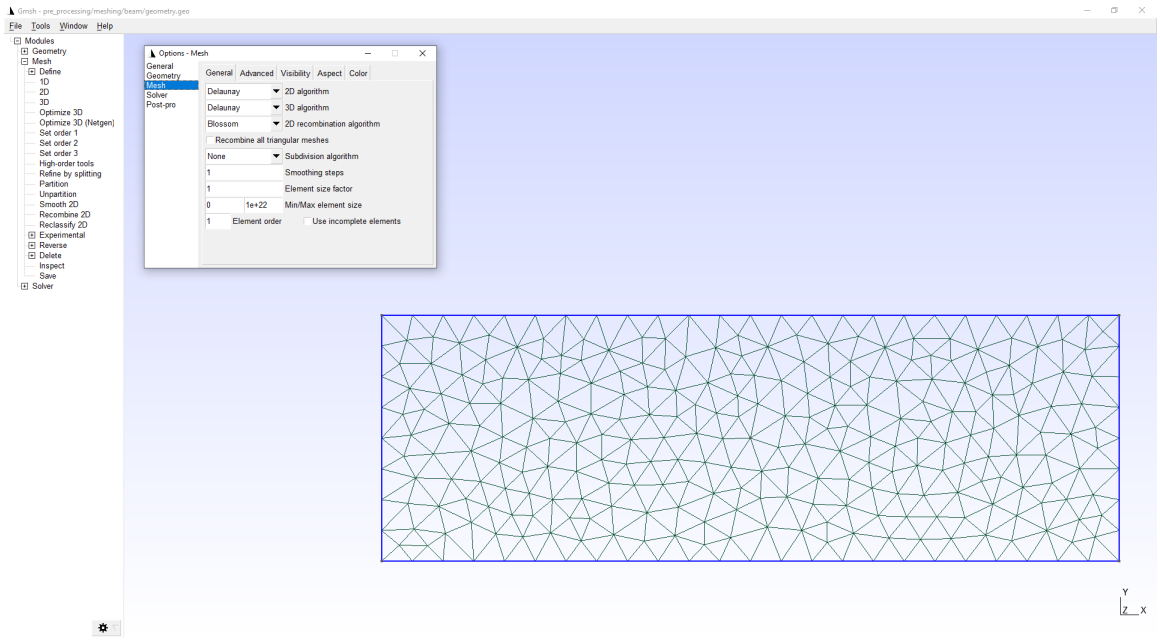
Figura 20 – Geometria criada para o problema da barragem.



Fonte: O autor (2021).

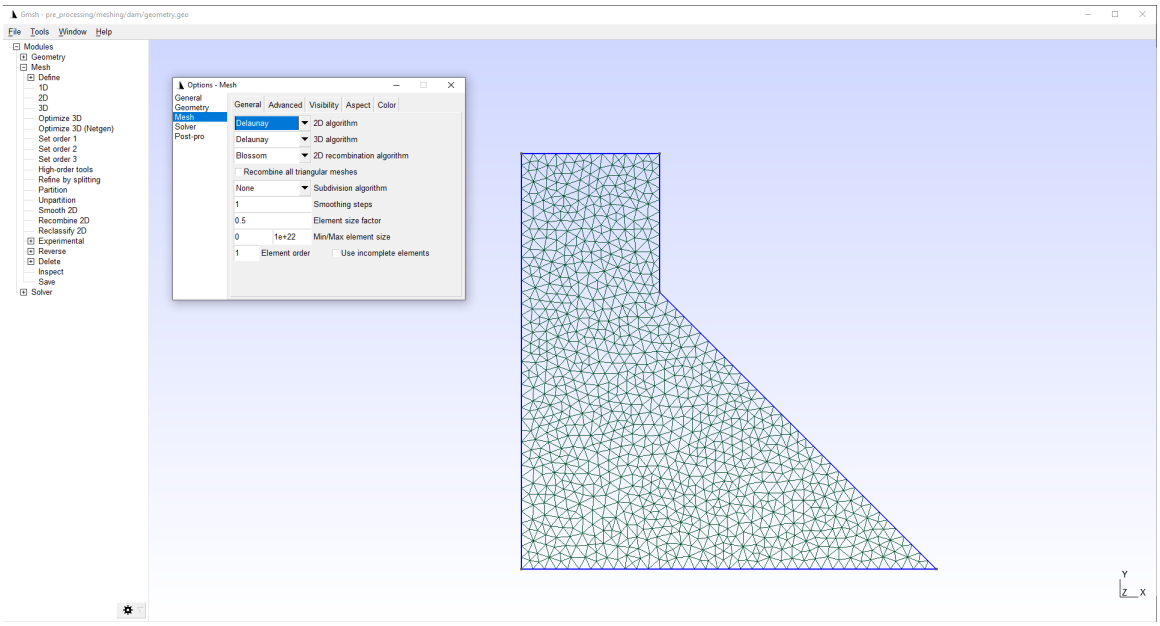
Com a geração da malha de elementos triangulares lineares, obtiveram-se as discretizações apresentadas nas Figuras 21 e 22, que também mostram as configurações de malha utilizadas. Nota-se uma distribuição uniforme dos 518 elementos para a viga parede e 1888 para a barragem.

Figura 21 – Geração da malha para a viga parede.



Fonte: O autor (2021).

Figura 22 – Geração da malha para a barragem.



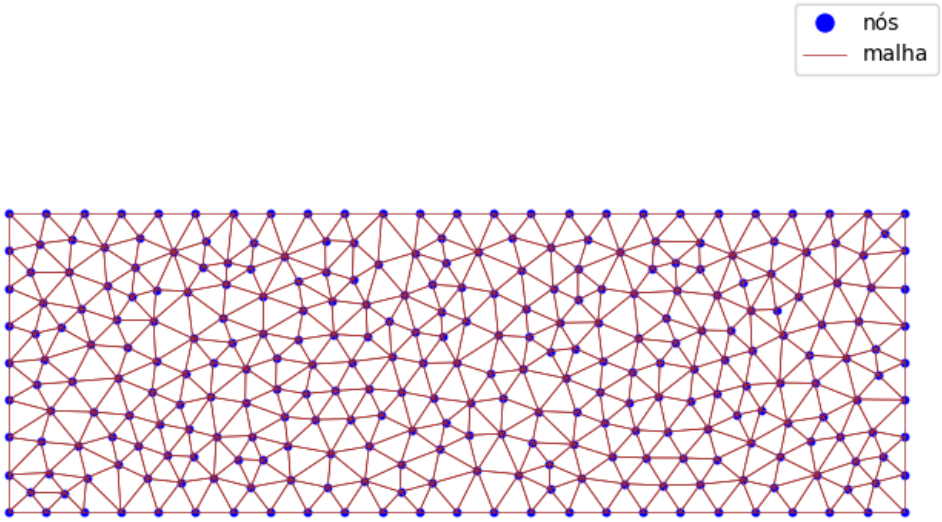
Fonte: O autor (2021).

Um ponto interessante do Gmsh é sua flexibilidade. Ao adicionar um ponto dentro da superfície da malha, o processamento leva em conta que ele deve deve ser tratado como um nó. Isso contribui para obter diretamente os deslocamentos desejados no centro da viga parede, por exemplo. O mesmo se aplica a curvas para o caso bidimensional. Uma forma de identificar esses nós, ou curvas, é o adicioná-los a um

grupo físico, facilmente filtrável no código em Python.

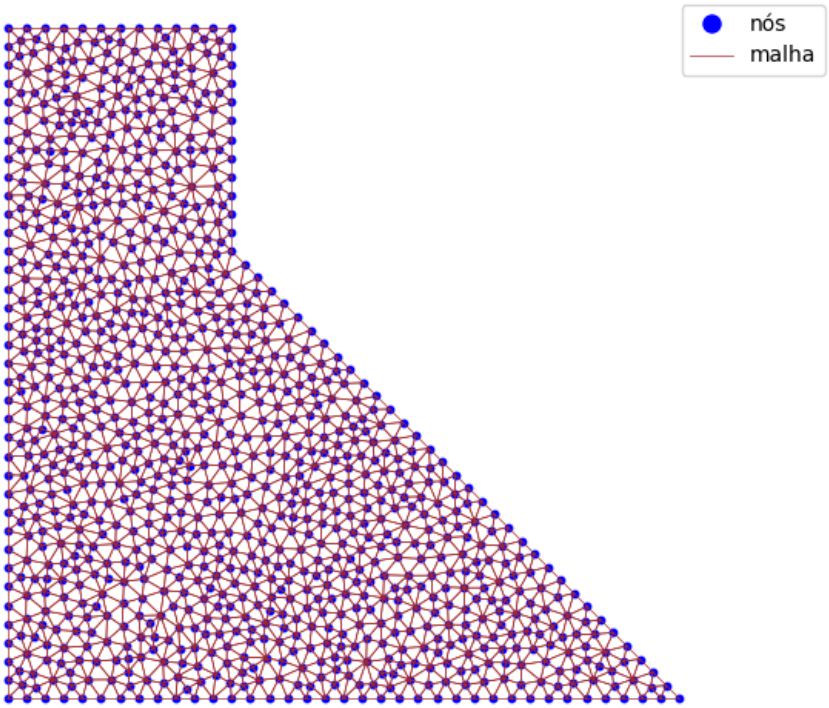
Uma vez gerado o .msh, verificou-se a eficácia da leitura do arquivo no Python por meio da plotagem de gráficos contendo os nós e os elementos. As Figuras 23 e 24 mostram que as malhas obtidas no Python são as mesmas representadas no Gmsh nas Figuras 21 e 22, respectivamente.

Figura 23 – Malha da viga parede lida no Python.



Fonte: O autor (2021).

Figura 24 – Malha da barragem lida no Python.

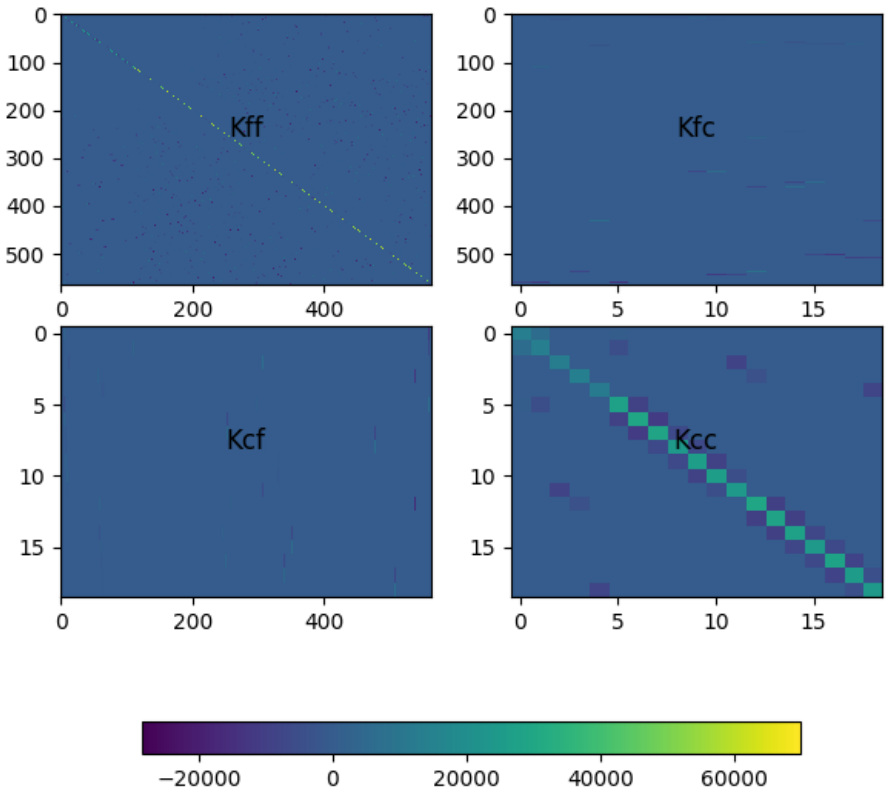


Fonte: O autor (2021).

4.3 MEF EM PYTHON

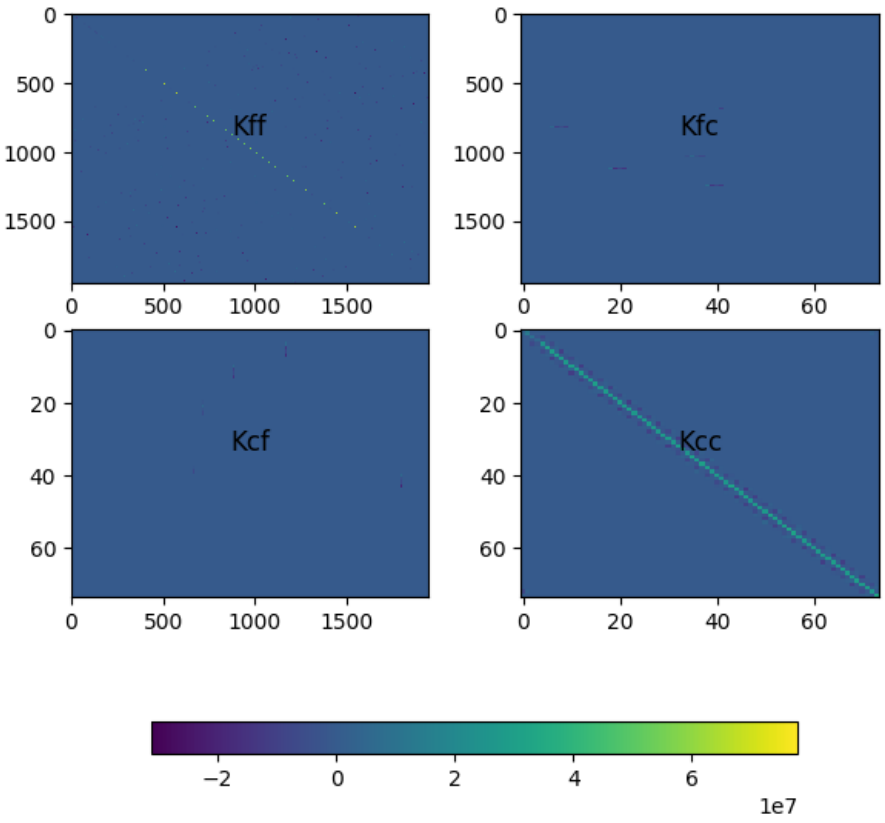
Para análise do algoritmo em Python, plotaram-se as partições da matriz de rigidez global, de modo a verificar a simetria e comportamento. As Figuras 25 e 26 mostram que as matrizes são simétricas.

Figura 25 – Partições da matriz de rigidez da viga parede.



Fonte: O autor (2021).

Figura 26 – Partições da matriz de rigidez da barragem.



Fonte: O autor (2021).

Obtidos os valores dos deslocamentos no processo de cálculo, pode-se fazer o pós-processamento do resultado de forma a visualizar a deformada obtida e verificar o comportamento das estruturas. Então, de maneira geral, os resultados do MEF são apresentados na seção de pós-processamento.

4.4 PÓS PROCESSAMENTO COM GMSH E PYTHON

Calculados os deslocamentos, as deformadas são apresentadas para visualização inicial do comportamento da estrutura e, por serem muito pequenas, foram exageradas por um fator adequado para visualização.

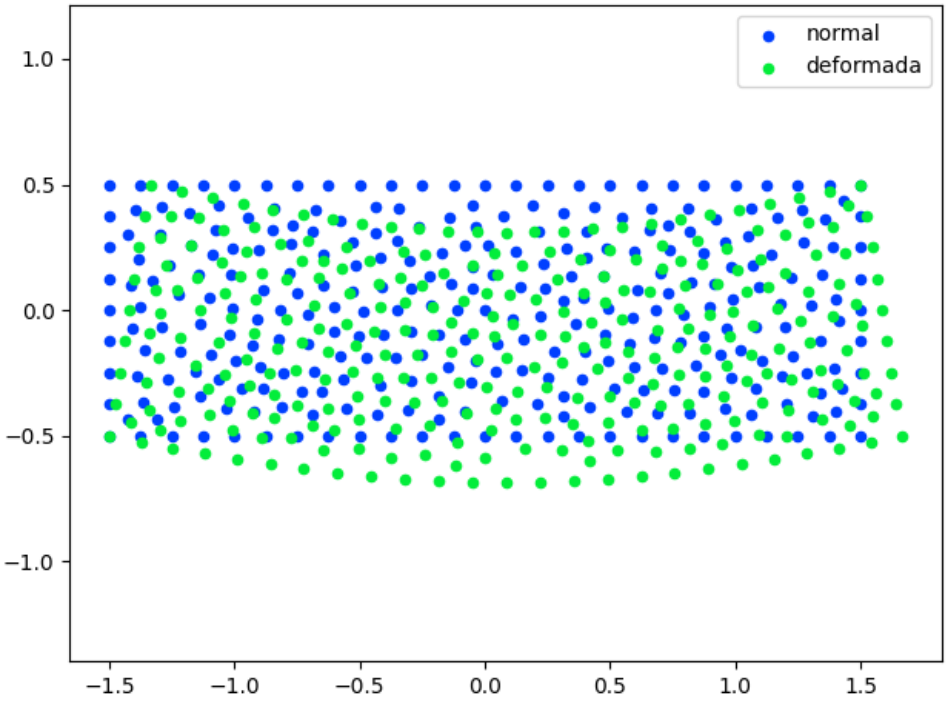
Uma vez escritas as tensões e deformações em um arquivo .msh, foi possível fazer a leitura no Gmsh e obter os gráficos com os valores calculados. É importante ressaltar que a etapa de visualização também pode ser feita por meio da biblioteca *matplotlib* do Python. Para demonstrar as duas opções, os resultados da viga parede foram pós processados em Python e o da barragem no Gmsh.

Nesta seção, serão separados os resultados obtidos para cada análise.

4.4.1 Viga parede

A Figura 27 mostra que a viga sofreu flexão com tração em sua parte inferior e compressão na superior. Nota-se, também, que as condições de contorno essenciais foram satisfeitas, uma vez que não houve deslocamentos em y dos nós nos apoios da viga e nem em x no nó totalmente fixado na coordenada (-0.5, -1.5).

Figura 27 – Deformada da viga parede (aumentada em 250 vezes).



Fonte: O autor (2021).

Conforme observa-se na Tabela 1, Almeida (2005) discretizou a viga parede com três malhas triangulares com diferentes números de elementos. A malha mais grosseira, com 16 elementos, gerou o erro relativo mais elevado, da ordem de 49% em relação ao deslocamento analítico, enquanto que para malha mais refinada, com 192 elementos, o erro relativo foi reduzido à ordem de 9%. A discretização com a malha de 518 elementos deste trabalho gerou um deslocamento vertical δ = 0,7702 mm, e um erro relativo em relação do deslocamento analítico de apenas 2,29%. Deste modo, os resultados obtidos neste trabalho, especialmente em relação ao deslocamento vertical, são compatíveis e muito próximos à resposta analítica e até mesmo semelhantes à resposta numérica calculada por Almeida (2005) com o uso de outro software de análise estrutural.

Tabela 1 – Comparação dos deslocamentos em y no centro da viga parede.

Solução	Malha	δ_y (mm)	Erro percentual relativo (%)
Timoshenko e Goodier (1951)	-	0,7931	-
Almeida (2005)	16 T3	0,4019	49,32
	96 T3	0,6532	17,63
	192 T3	0,7155	9,78
O autor (2021)	518 T3	0,7702	2,29

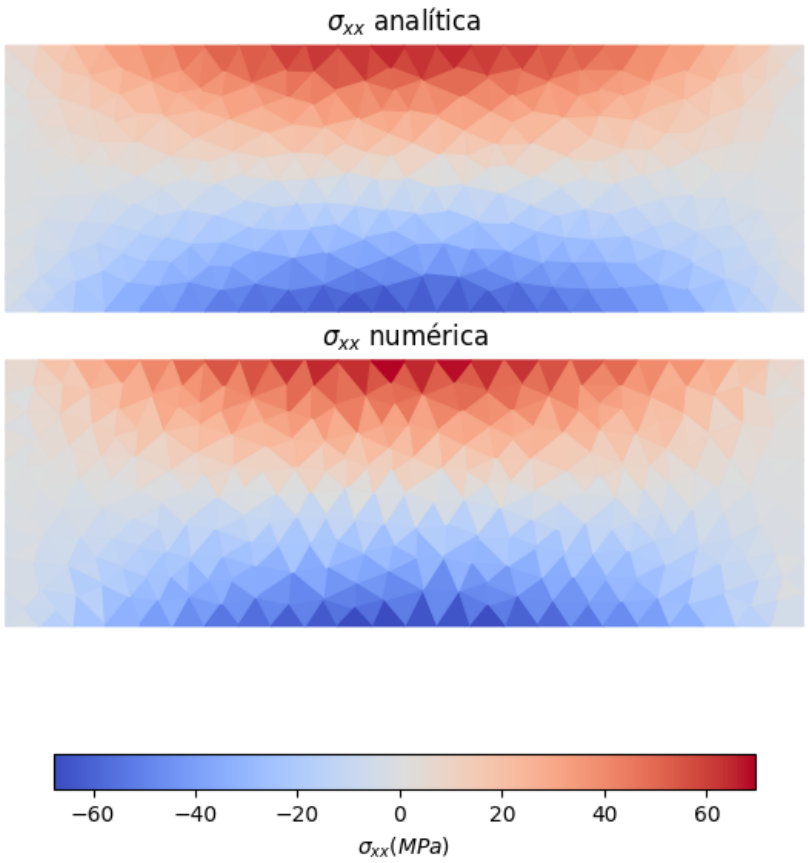
A Figura 28 mostra as isofaixas das tensões σ_{xx} obtidas para a viga parede. Assim, a cor vermelha representa a compressão e a cor azul tração. Observa-se coerência em relação à deformada da Figura 27, visto que a parte inferior da viga está tracionada e a parte superior está comprimida. De fato, esse é o comportamento esperado para uma viga biapoiada com o carregamento imposto definido no problema.

A solução analítica para o problema pode ser encontrada em Timoshenko e Goodier (1951), onde σ_{xx} é calculado pela Equação 31.

$$\sigma_{xx} = \frac{q}{2I}(L^2 - x^2)y + \frac{q}{2I}\left(\frac{2}{3}y^3 - \frac{2}{5}c^2 y\right) \tag{31}$$

em que L é o comprimento da viga, c a altura sobre dois e I o momento de inércia. Para x e y utilizaram-se as coordenadas dos centróides dos elementos triangulares. A Figura 28 mostra o comparativo da solução analítica com a numérica.

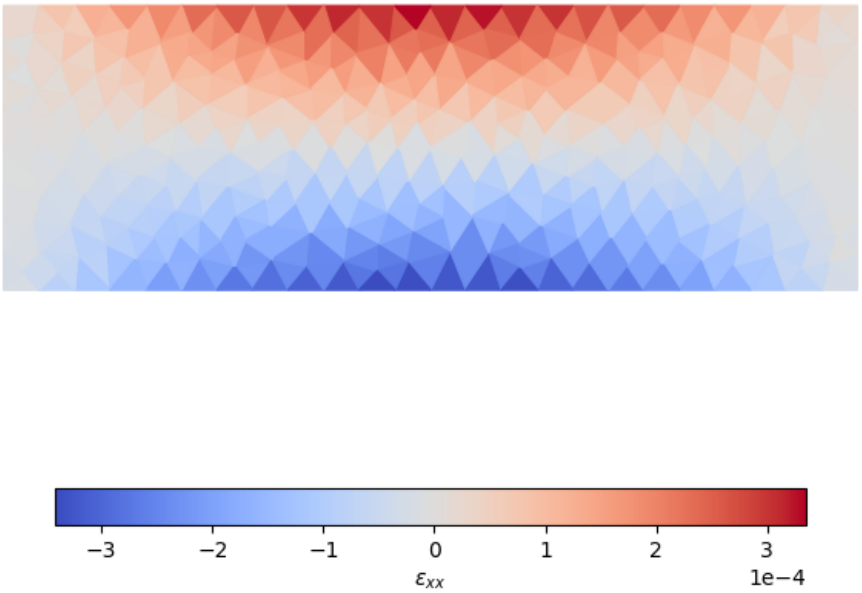
Figura 28 – Isofaixas de tensão σ_{xx} numérica e analítica.



Fonte: O autor (2021).

As deformações correspondentes ε_{xx} podem ser visualizadas na Figura 29. Nota-se a compatibilidade com as tensões atuantes σ_{xx}, em uma relação diretamente proporcional.

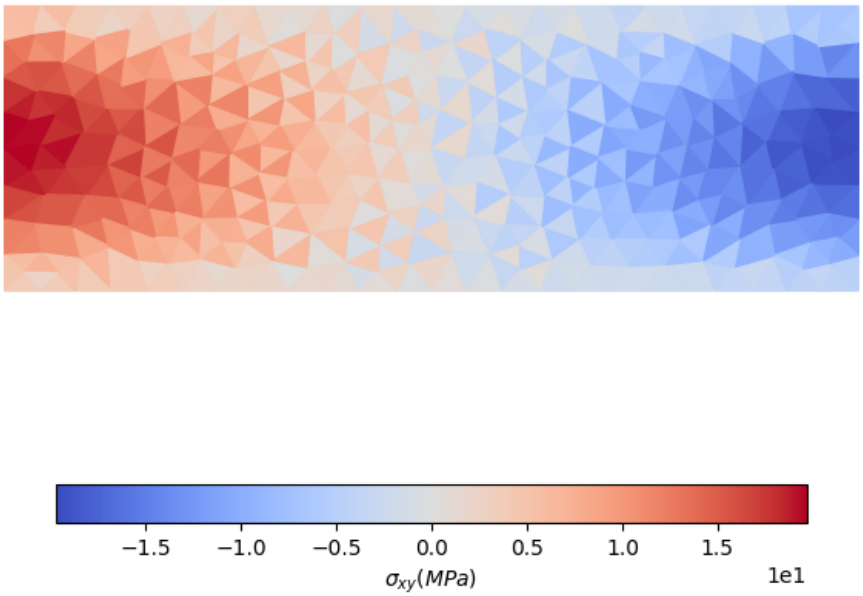
Figura 29 – Isofaixa de deformação ε_{xx}.



Fonte: O autor (2021).

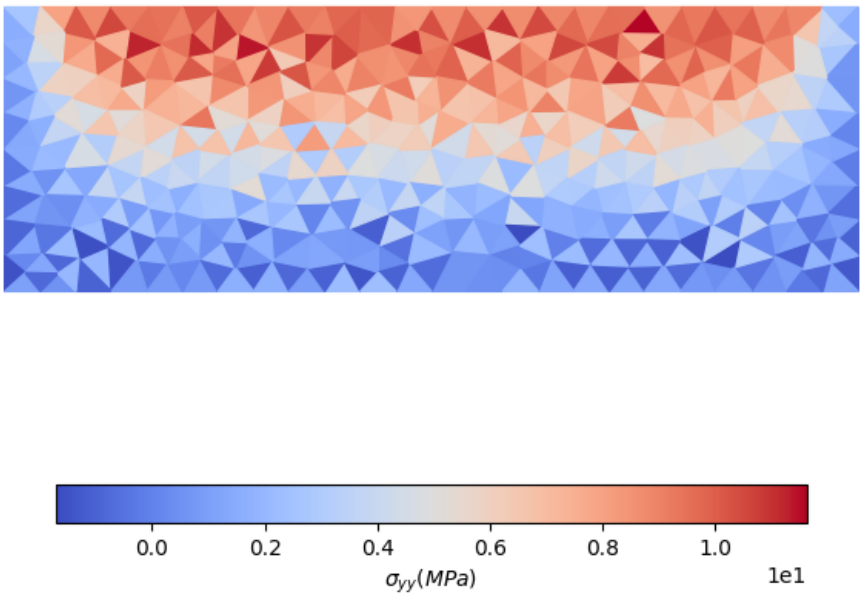
A Figura 30 mostra as tensões cisalhantes. Observa-se que as tensões são maiores próximo aos apoios, onde, de fato, os esforços cortantes são elevados. Já a Figura 31 traz as tensões σ_{yy}. A solução de Timoshenko e Goodier (1951) considera que a viga sofre flexão pura, em que as tensões σ_{yy} são consideradas nulas. De fato, as tensões obtidas pela simulação são consideravelmente inferiores, o que corrobora para a validação do algoritmo.

Figura 30 – Isofaixa de tensão σ_{xy} (*MPa*).



Fonte: O autor (2021).

Figura 31 – Isofaixa de tensão σ_{yy} (*MPa*).

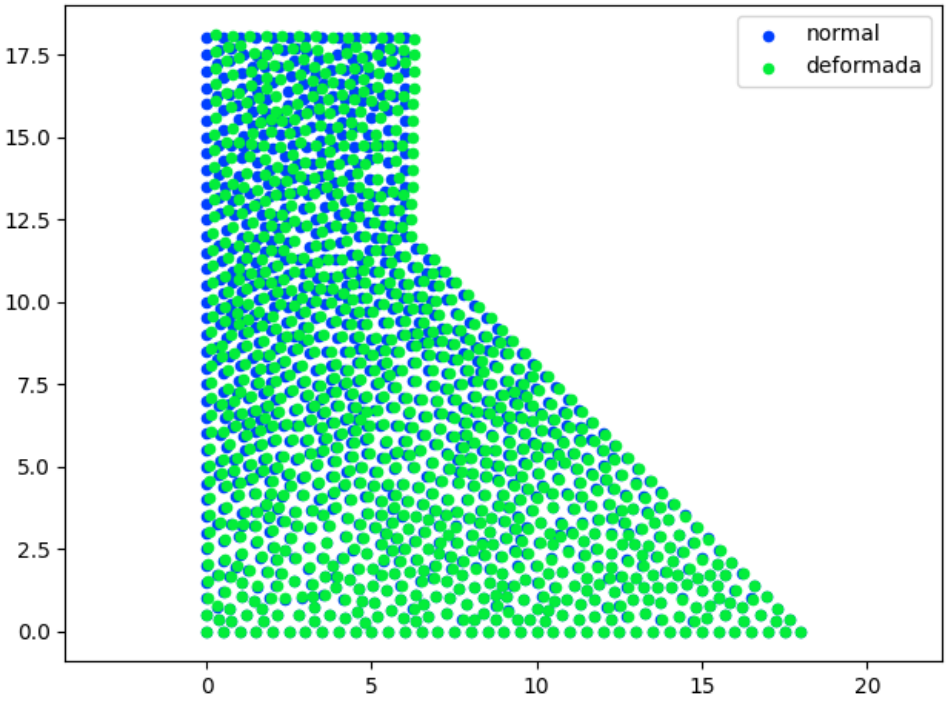


Fonte: O autor (2021).

4.4.2 Barragem

A deformada da barragem é ilustrada na Figura 32. Nota-se que a lateral submetida a carregamento sofre deslocamento na direção esperada. Ainda, de maneira análoga à viga parede, as condições de contorno essenciais foram satisfeitas (todos os nós da base da barragem são fixos).

Figura 32 – Deformada da barragem (aumentada em 1000 vezes).

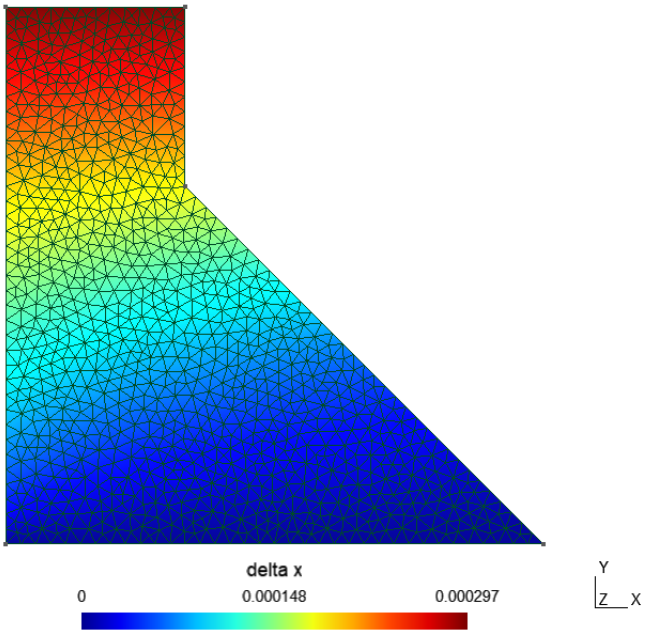


Fonte: O autor (2021).

Os deslocamentos horizontais e verticais obtidos nas direções x e y podem ser visualizados nas Figuras 33 e 34, respectivamente. Nota-se que os maiores deslocamentos estão localizados no ponto A (ver Figura 18), com valor de $2{,}97 \times 10^{-4}$m na direção x e de $9{,}81 \times 10^{-5}$m na direção y. Na Tabela 2 são escritos os deslocamentos obtidos no ponto A obtidos neste trabalho e por Almeida (2005) para as suas quatro malhas de elementos finitos. É importante ressaltar que Almeida (2005) adota elementos triangulares de seis nós (T6), que possuem interpolação quadrática e são classificados como elementos de segunda ordem. O autor observou que os deslocamentos horizontais e verticais convergiram ao refinar as malhas de elementos T6, conforme mostra a Tabela 2. O elemento triangular de três nós aqui utilizado tem interpolação linear, e é classificado como elemento de primeira ordem. Assim, a diferença entre os elementos triangulares de três e seis nós justifica o porquê dos valores de deslocamentos verticais e horizontais não terem convergido para os valores

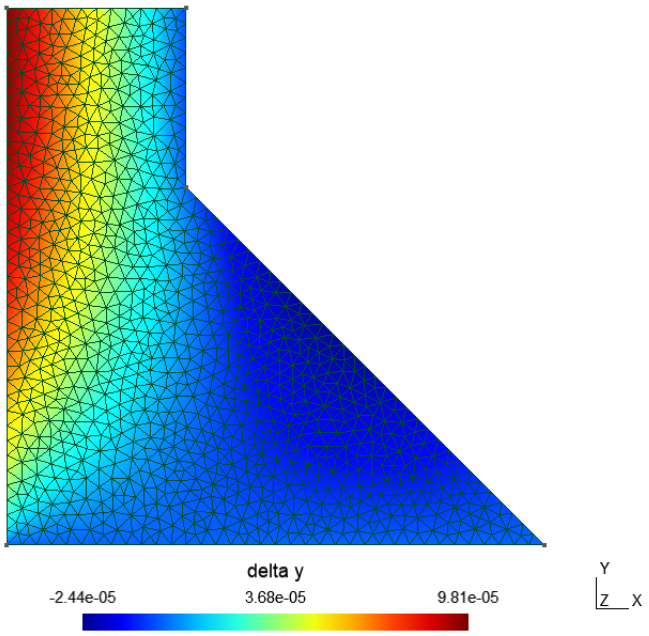
obtidos por Almeida (2005) com a malha mais refinada. Os resultados aqui obtidos foram muito semelhantes aos de Almeida (2005), assim como o comportamento estrutural da barragem.

Figura 33 – Deslocamentos horizontais δ_x.



Fonte: O autor (2021).

Figura 34 – Deslocamentos δ_y.



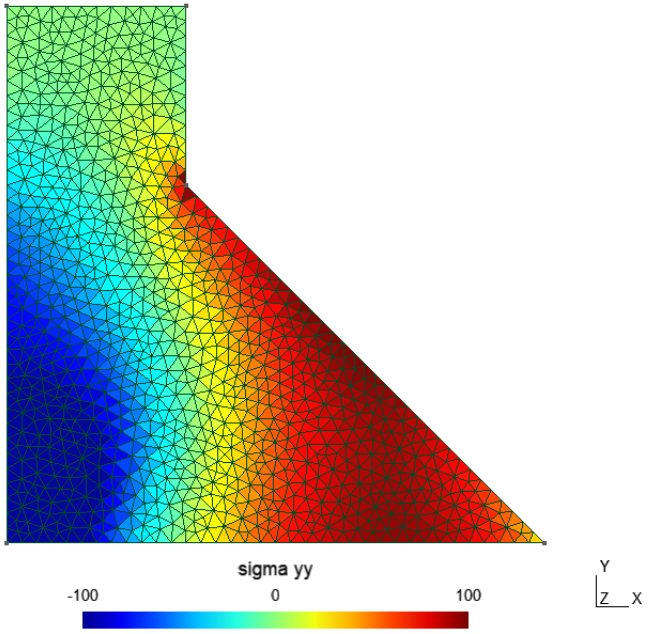
Fonte: O autor (2021).

Tabela 2 – Comparação dos deslocamentos no ponto A da barragem.

Solução	Malha	δ_x (mm)	δ_y (mm)
Almeida (2005)	16 T6	0,2993	0,0992
	90 T6	0,3041	0,1009
	360 T6	0,3050	0,1012
	108 Q8 e 144 T6	0,3050	0,1012
O autor (2021)	1888 T3	0,2970	0,0981

A Figura 35 mostra as isofaixas de tensão σ_{yy}. Nota-se, de maneira análoga à explicada para a viga parede, que a região trapezoidal da barragem possui tensões positivas, indicando compressão (região vermelha). Já para a face a montante, tem-se tensões negativas, indicando tração. Tendo em vista que a combinação de carga com o engaste da base tende a fletir a barragem à jusante, faz sentido essa distribuição de tensão.

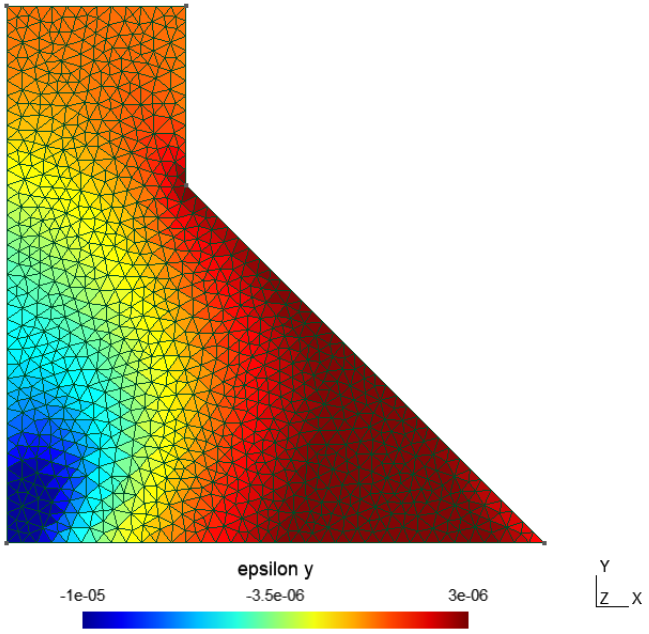
Figura 35 – Isofaixas de tensão σ_{yy} (*kPa*).



Fonte: O autor (2021).

Ao analisar o gráfico de deformações ε_{yy} mostrado na Figura 36, nota-se a compatibilidade tensão-deformação, de maneira análoga à viga parede.

Figura 36 – Deformações ε_{yy}.



Fonte: O autor (2021).

Com aplicação de ambos os modelos de análise EPT e EPD foi possível reproduzir os comportamentos estruturais de elementos que foram analisados numericamente por Almeida (2005) e analiticamente por Timoshenko e Goodier (1951).

É possível melhorar os resultados por meio da implementação de outras formas de discretização, como fez Almeida (2005). No entanto, como o trabalho objetiva introduzir o MEF, utilizar a discretização de elementos triangulares com funções de forma lineares é pertinente. A simplicidade aliada com bons resultados fazem dessa abordagem a mais adequada em um contexto introdutório.

5 CONCLUSÕES

O trabalho de conclusão de curso aqui apresentado teve como objetivo principal apresentar e aplicar o MEF para solucionar problemas de EPT e EPD de maneira prática e didática. Assim, este trabalho propiciou as seguintes constatações:

- o Gmsh é uma ferramenta interessante para pré e pós-processamento de malhas, tendo em vista que supriu as necessidades deste trabalho;
- o Python possui diversas ferramentas que auxiliam na solução de problemas numéricos via MEF, como as bibliotecas *numpy* e *scipy* para tratamento numérico e até mesmo *matplotlib* para visualização;
- os equacionamentos necessários para aplicação básica do MEF foram levantados;
- os resultados obtidos neste trabalho, validados por dois exemplos para os modelos de análise EPT e EPD, foram possíveis em razão do algoritmo desenvolvido e da união entre os recursos tecnológicos Gmsh e Python;
- apesar de constituir a discretização mais simples para problemas bidimensionais, os elementos triangulares de três nós com funções de forma lineares apresentaram bons resultados as discretizações utilizadas;
- foi possível desenvolver um material de consulta introdutório ao MEF para a comunidade acadêmica que possa se interessar pelo método;
- como propostas de trabalhos futuros, é possível analisar os problemas aqui discutidos do ponto de vista de refinamento de malha e implementar o elemento finito triangular de seis nós.

REFERÊNCIAS

ALMEIDA, M. L. de. **Elementos Finitos Paramétricos Implementados em Java**. Dissertação (Mestrado em Engenharia de Estruturas) — Escola de Engenharia, Universidade Federal de Minas Gerais, 2005.

BRENNER, S. C.; SCOTT, L. R. **The mathematical theory of finite element methods**. 3. ed. Heidelberg: Springer, 2008. 405 p.

GEUZAINE, C.; REMACLE, J.-F. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. **INT J NUMER METH ENG**, p. 1–24, 2009.

GONÇALVES, M. A. B. **Geração de malhas bidimensionais de elementos finitos baseada em mapeamentos transfinitos**. Dissertação (Mestrado em Engenharia de Estruturas) — Escola de Engenharia, Universidade Federal de Minas Gerais, 2004.

KARASUDHI, P. **Foundations of solid mechanics**. Heidelberg: Springer, 1991. v. 3. 446 p.

LIU, G. R.; QUEK, S. S. **The finite element method**: a pratical course. Oxford: Butterworth-Heinemann, 2003. 384 p.

MOREIRA, R. N. **Sistema gráfico interativo para ensino de análise estrutural através do método dos elementos finitos**. Dissertação (Mestrado em Engenharia de Estruturas) — Escola de Engenharia, Universidade Federal de Minas Gerais, 2006.

OÑATE, E. **Structural Analysis with the Finite Element Method**: Linear statics. Heidelberg: Springer, 2009. 472 p.

TIMOSHENKO, S.; GOODIER, J. N. **Theory of Elasticity**. New York: McGRAW-HILL BOOK COMPANY, 1951. 506 p.

ZANELLA, L. C. H. **Metodologia de pesquisa**. 2. ed. Florianópolis: Centro Socioeconômico, Universidade Federal de Santa Catarina, 2013.