

images/logo\_ubi\_vprincipal

## **Trabalho 03 – Placas de Kirchhoff: Soluções Numéricas**

**José Miguel Correia Barros, M15321**

**Engenharia Aeronáutica**  
(2º ciclo de estudos)

Docente: Prof. Doutor Thiago Assis Dutra

**24 de Novembro de 2025**

# Índice

|                                      |            |
|--------------------------------------|------------|
| <b>Índice</b>                        | <b>i</b>   |
| <b>Lista de Figuras</b>              | <b>iii</b> |
| <b>Lista de Tabelas</b>              | <b>iv</b>  |
| <b>1 Descrição do Problema</b>       | <b>1</b>   |
| 1.1 Tarefa 1 . . . . .               | 1          |
| 1.2 Tarefa 2 . . . . .               | 2          |
| <b>2 Metodologia de Solução</b>      | <b>4</b>   |
| 2.1 Tarefa 1 . . . . .               | 4          |
| 2.2 Tarefa 2 . . . . .               | 6          |
| 2.2.1 Alínea a) . . . . .            | 6          |
| 2.2.2 Alínea b) . . . . .            | 6          |
| <b>3 Implementação do código</b>     | <b>10</b>  |
| <b>4 Resultados e Discussões</b>     | <b>12</b>  |
| 4.1 Resultados da Tarefa 1 . . . . . | 12         |
| 4.2 Resultados da Tarefa 2 . . . . . | 13         |
| 4.2.1 Alinea a) . . . . .            | 13         |
| 4.2.2 Alinea b) . . . . .            | 13         |
| 4.2.3 Alinea c), caso (i) . . . . .  | 16         |
| 4.2.4 Alinea c), caso (ii) . . . . . | 17         |
| <b>5 Conclusão</b>                   | <b>18</b>  |
| <b>Bibliografia</b>                  | <b>19</b>  |
| <b>A Implementação do código</b>     | <b>20</b>  |

|                        |    |
|------------------------|----|
| A.1 Tarefa 2 . . . . . | 24 |
|------------------------|----|

# **Lista de Figuras**

|     |  |    |
|-----|--|----|
| 1.1 | Placa ortotrópica referente à Tarefa 1 . . . . .                             | 1  |
| 1.2 | Placa ortotrópica referente à Tarefa 2 . . . . .                             | 3  |
| 3.1 | Fluxograma referente á Tarefa 1 . . . . .                                    | 10 |
| 3.2 | Fluxograma referente à Tarefa 2 . . . . .                                    | 10 |
| 4.1 | Primeiro Modo de Vibração . . . . .  | 12 |
| 4.2 | Deflexões na placa considerando uma discretização em 411 elementos. . . . .  | 14 |
| 4.3 | Distribuição das reações e momentos na zona de encastramento. . . . .        | 14 |
| 4.4 | Distribuição de tensões $\sigma_x$ na superfície superior. . . . .           | 15 |
| 4.5 | Distribuição de tensões $\sigma_y$ na superfície superior. . . . .           | 15 |
| 4.6 | Distribuição de tensões de corte $\tau_{xy}$ na superfície superior. . . . . | 16 |

# **Lista de Tabelas**

|   |   |
|---|---|
| 1.1 Propriedades geométricas, mecânicas e condições de fronteira. . . . . | 3 |
|---|---|

# **Capítulo 1**

## **Descrição do Problema**

O presente relatório apresenta a proposta de resolução do Trabalho 3 da Unidade Curricular de Placas e Cascas, focando na análise de placas finas ortotrópicas submetidas a carregamentos externos, descritas pela equação de Kirchhoff. Este modelo é fundamental um vez que permite determinar a deflexão das placas sob forças aplicadas. Como soluções analíticas são muitas vezes impraticáveis, recorrem-se a métodos numéricos, sendo o Método dos Elementos Finitos amplamente utilizado para obter resultados precisos em problemas estruturais complexos.

### **1.1 Tarefa 1**

Na Tarefa 1 é analisada uma placa com dimensões  $a$  e  $b$ , espessura  $t$ , composta por um material ototrópico com massa volúmica  $\rho$ . Esta placa encontra-se encastrada nas extremidades correspondentes a  $x = 0$  e  $x = a$ , enquanto nas bordas  $y = 0$  e  $y = b$  está simplesmente apoiada. A configuração geométrica e as condições de fronteira da placa podem ser observadas na figura 1.1,

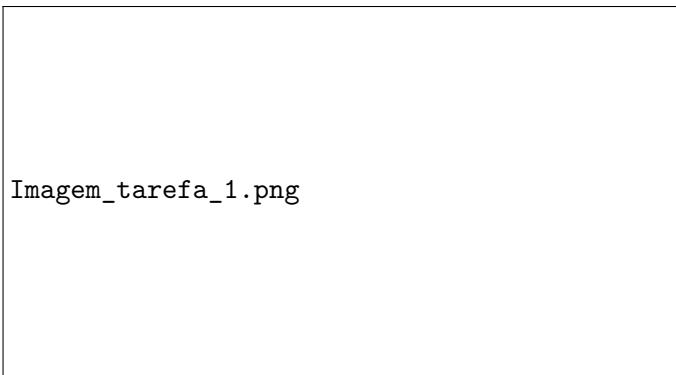


Figura 1.1: Placa ortotrópica referente à Tarefa 1.

Na condição de vibração livre, a equação da energia potencial para a placa é dada por (1.1):

$$\begin{aligned}\Pi = \frac{1}{2} \int \int & \left[ D_{11} (w_{,xx})^2 + 2D_{12} w_{,xx} w_{,yy} \right. \\ & \left. + D_{22} (w_{,yy})^2 + 4D_{66} (w_{,xy})^2 \right] dy dx \quad (1.1) \\ & - \frac{1}{2} \omega^2 \rho t \int \int w^2 dy dx = 0\end{aligned}$$

Através do Método de Ritz, vai-se determinar frequência fundamental  $\omega$  associada à vibração livre da placa ilustrada na figura 1.1, bem como o respetivo modo de vibração. Para tal, considera-se que a forma aproximada da deflexão da placa retangular pode ser expressa conforme apresentado na equação (1.2), de acordo com a formulação do Método de Ritz.

$$w(x, y) \approx W_{mn}(x, y) = \sum_{i=1}^M \sum_{j=1}^N c_{ij} X_i(x) Y_j(y) \quad (1.2)$$

Onde  $c_{ij}$ , corresponde ao coeficiente de Ritz e  $X_i(x)$  e  $Y_j(y)$  devem cumprir apenas as condições de fronteira geométricas impostas ao problema.

Em  $x = 0$  e  $x = a$  a placa da figura 1.1 encontra-se encastradas nessas bordas e por isso  $X_i(x)$  é dado pela equação (1.3):

$$X_i(x) = \left( \frac{x}{a} \right)^{i+1} - 2 \left( \frac{x}{a} \right)^{i+2} + \left( \frac{x}{a} \right)^{i+3} \quad (1.3)$$

De maneira análoga, em  $y = 0$  e  $y = b$  a placa encontra-se simplesmente apoiada e por isso  $Y_j(y)$  é dada pela equação (1.4):

$$Y_j(y) = \left( \frac{y}{b} \right)^j - \left( \frac{y}{b} \right)^{j+1} \quad (1.4)$$

## 1.2 Tarefa 2

Uma placa fina de dimensões  $a \times b$  e espessura  $t$ , feita de um material ortotrópico, está sujeita a uma pressão  $p_0$  uniformemente distribuída sobre toda a sua superfície superior (plano  $xy$ ). As arestas da placa são designadas por Borda 1, Borda 2, Borda 3 e Borda 4 conforme a figura 1.2.

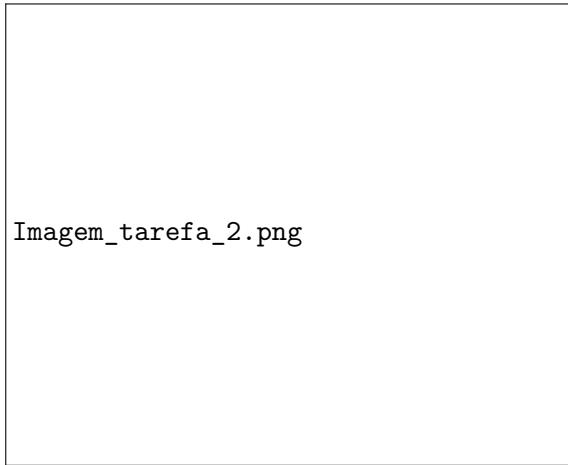


Figura 1.2: Placa ortotrópica referente à Tarefa 2

Na alínea *a*) é solicitada a determinação da expressão da matriz de elasticidade  $[D]$ , assumindo que os eixos principais de ortotropia do material estão orientados paralelamente aos eixos  $x$  e  $y$ .

Na alínea *b*), pretende-se o desenvolvimento de um programa de cálculo baseado no Método dos Elementos Finitos (FEM), capaz de determinar as deflexões  $w(x, y)$  da placa no domínio definido por  $x \in [0, a]$  e  $y \in [0, b]$ , bem como de avaliar as forças de reacção nos apoios correspondentes.

Na alínea *c*), é pedido que se analisem dois casos particulares:

- (i) Quando os eixos principais do material não estão alinhados com os eixos  $x$  e  $y$ ;
- (ii) Quando o material é constituído por um compósito laminado simétrico, não balanceado e composto por  $n$  camadas.

Adicionalmente, são fornecidas algumas notas complementares:

- (i) Na discretização da placa, podem ser utilizados elementos triangulares de três nós (com 9 graus de liberdade) ou elementos quadriláteros de quatro nós (com 12 graus de liberdade);
- (ii) A malha deve ser progressivamente refinada até se alcançar a convergência dos resultados, devendo a escolha final do número de elementos ser devidamente justificada.

Os parâmetros necessários para o desenvolvimento da Tarefa 1 e para a Tarefa 2 são apresentados na Tabela 1.1.

| Caso | $a$ [mm] | $b$ [mm] | $t$ [mm] | $E_1$ [GPa] | $E_2$ [GPa] | $\nu_{12}$ | $G_{12}$ [GPa] | $p_0$ [N/m <sup>2</sup> ] | B1 | B2 | B3 | B4 |
|------|----------|----------|----------|-------------|-------------|------------|----------------|---------------------------|----|----|----|----|
| 6    | 750      | 600      | 2        | 130         | 10          | 0.26       | 5              | 175                       | C  | C  | F  | F  |

Tabela 1.1: Propriedades geométricas, mecânicas e condições de fronteira.

# Capítulo 2

## Metodologia de Solução

### 2.1 Tarefa 1

Para determinar a frequência fundamental de vibração  $\omega$  e o modo de vibração correspondente de uma placa ortotrópica, aplica-se o método de aproximação de Ritz, conforme indicado na equação (1.2). As funções  $X_i(x)$  e  $Y_j(y)$  são escolhidas de modo a satisfazer as condições de fronteira do problema. A partir desta aproximação, podem obter-se as derivadas de  $w(x, y)$ , designadamente  $w_{,xx}$ ,  $w_{,yy}$  e  $w_{,xy}$ , que são posteriormente substituídas na equação de equilíbrio correspondente ao caso ortotrópico.

No caso ortotrópico, os momentos fletores e as curvaturas estão relacionados pela matriz de rigidez flexional  $[D]$ , expressa pela equação (2.1):

$$\begin{bmatrix} M_x \\ M_y \\ M_{xy} \end{bmatrix} = [D] \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix}. \quad (2.1)$$

A matriz  $[D]$  representa a rigidez à flexão da placa e é dada por:

$$[D] = \frac{t^3}{12} \begin{bmatrix} Q_{11} & Q_{12} & 0 \\ Q_{12} & Q_{22} & 0 \\ 0 & 0 & Q_{66} \end{bmatrix}. \quad (2.2)$$

Os coeficientes  $Q_{11}$ ,  $Q_{22}$ ,  $Q_{12}$  e  $Q_{66}$  dependem das propriedades elásticas do material e são definidos pelas equações (2.3) a (2.6):

$$Q_{11} = \frac{E_1}{1 - \nu_{12}\nu_{21}}, \quad (2.3)$$

$$Q_{22} = \frac{E_2}{1 - \nu_{12}\nu_{21}}, \quad (2.4)$$

$$Q_{12} = \frac{\nu_{12}E_2}{1 - \nu_{12}\nu_{21}}, \quad (2.5)$$

$$Q_{66} = G_{12}. \quad (2.6)$$

A variação da energia potencial pode ser escrita como:

$$\delta\Pi = \sum_{i=1}^m \sum_{j=1}^n \frac{\partial\Pi}{\partial c_{ij}} \delta c_{ij}, \quad (2.7)$$

sendo  $c_{ij}$  os coeficientes da função aproximada  $w(x, y)$  utilizada no método de Ritz.

A partir da equação (2.7), obtém-se as relações apresentadas nas equações (2.8) a (2.11), que exprimem as variações das derivadas parciais de  $w$  relativamente aos coeficientes  $c_{ij}$ :

$$\delta w_{,xx} = \frac{\partial w_{,xx}}{\partial c_{ij}} \delta c_{ij}, \quad (2.8)$$

$$\delta w_{,yy} = \frac{\partial w_{,yy}}{\partial c_{ij}} \delta c_{ij}, \quad (2.9)$$

$$\delta w_{,xy} = \frac{\partial w_{,xy}}{\partial c_{ij}} \delta c_{ij}, \quad (2.10)$$

$$\delta w = \frac{\partial w}{\partial c_{ij}} \delta c_{ij}. \quad (2.11)$$

Substituindo as equações (2.8)–(2.11) na expressão da energia potencial (2.7), e considerando a matriz  $[D]$  definida pela matriz (2.2), obtemos a equação de equilíbrio da placa ortotrópica.

Como as variações  $\delta$  são arbitrárias, o termo que as acompanha deve anular-se. Assim, ao reorganizar a equação em função dos coeficientes  $c_{ij}$ , obtém-se a equação característica da placa, que permite determinar a frequência fundamental de vibração  $\omega$ .

Com base nesta equação, é possível obter o modo de vibração associado. Para representar graficamente este modo, pode ser utilizado um programa desenvolvido em linguagem *Python*, que gera a superfície  $w(x, y)$  considerando valores unitários para  $a$ ,  $b$  e  $c_{ij}$ . Esta representação permite visualizar a forma modal correspondente à frequência fundamental da placa ortotrópica.

## 2.2 Tarefa 2

Nesta tarefa, pretende-se determinar, para uma placa ortotrópica sujeita a um carregamento uniforme, a matriz de elasticidade  $[D]$ , as deflexões  $w(x, y)$ , as reações nos apoios e as tensões  $\sigma_x$ ,  $\sigma_y$  e  $\tau_{xy}$ , recorrendo ao Método dos Elementos Finitos (MEF).

### 2.2.1 Alínea a)

A matriz de rigidez flexional do material ortotrópico considerada nesta análise corresponde à obtida na Tarefa 1 e é dada pela equação (2.2). Esta matriz traduz a rigidez à flexão da placa e contém os coeficientes  $Q_{ij}$  definidos nas Equações (2.3)–(2.6), os quais dependem diretamente das propriedades elásticas do material ( $E_1$ ,  $E_2$ ,  $\nu_{12}$ ,  $\nu_{21}$  e  $G_{12}$ ).

### 2.2.2 Alínea b)

Nesta alínea, recorre-se ao Método dos Elementos Finitos (MEF) para determinar as deflexões e as reações na placa ortotrópica. O procedimento geral pode ser descrito pelos seguintes passos:

1. Determinação da matriz de rigidez elementar  $[K]^e$  de acordo com as propriedades do material e geometria do elemento;
2. Montagem da matriz global  $[K]$  a partir das contribuições de todos os elementos;
3. Cálculo do vetor global de forças nodais  $\{Q\}$  associado ao carregamento uniforme  $p_0$ ;
4. Aplicação das condições de fronteira e resolução do sistema linear de equações:

$$\{\delta\} = [K]^{-1}\{Q\};$$

5. Cálculo das reações nos apoios, obtidas por:

$$\{R\} = [K]_{\text{global}}\{\delta\} - \{Q\}.$$

A matriz de rigidez elementar é obtida pela Equação (2.12):

$$[K]^e = \left( [C]^{-1} \right)^T \left( \iint_A [H]^T [D] [H] dx dy \right) [C]^{-1}, \quad (2.12)$$

onde:

- $[H]$  é a matriz das derivadas das funções de forma do elemento;
- $[C]$  relaciona os coeficientes polinomiais  $a_i$  com os graus de liberdade nodais.

A matriz  $[H]^T$ , conforme apresentada em (2.13) é dada por:

$$[H]^T = \begin{bmatrix} 0 & 0 & 0 & -2 & 0 & 0 & -6x & -2y & 0 & 0 & -6xy & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & -2x & -6y & 0 & -6xy \\ 0 & 0 & 0 & 0 & -2 & 0 & 0 & -4x & -4y & 0 & -12x & -12y \end{bmatrix} \quad (2.13)$$

A equação (2.14) define a função de deslocamento  $w_e$  como:

$$w_e = a_1 + a_2x + a_3y + a_4x^2 + a_5xy + a_6y^2 + a_7x^3 + a_8x^2y + a_9xy^2 + a_{10}y^3 + a_{11}x^3y + a_{12}xy^3 \quad (2.14)$$

Desta forma, a matriz  $[C]$  é dada por (2.15):

$$[C] = \begin{bmatrix} \frac{\partial w_{ei}}{\partial a_1} & \dots & \frac{\partial w_{ei}}{\partial a_{12}} \\ \frac{\partial w_{ei}}{\partial \theta_{xi}} & \dots & \frac{\partial w_{ei}}{\partial \theta_{xi}} \\ \frac{\partial w_{ei}}{\partial a_1} & \dots & \frac{\partial w_{ei}}{\partial a_{12}} \\ \frac{\partial w_{ei}}{\partial \theta_{yi}} & \dots & \frac{\partial w_{ei}}{\partial \theta_{yi}} \\ \frac{\partial w_{ei}}{\partial a_1} & \dots & \frac{\partial w_{ei}}{\partial a_{12}} \\ \vdots & \ddots & \vdots \\ \frac{\partial w_{ei}}{\partial a_1} & \dots & \frac{\partial w_{ei}}{\partial a_{12}} \end{bmatrix} \quad (2.15)$$

Após a obtenção da matriz  $[C]$ , procede-se à sua inversão. De acordo com a equação (2.12), é então possível determinar a matriz  $[K]_e$  correspondente a cada elemento. O passo seguinte consiste em formar a matriz global  $[K]$ , a qual é construída através do processo de *meshing* das matrizes elementares  $[K]_e$ . Esse espalhamento é realizado com base na matriz de conectividade, que assegura a correta correspondência entre os graus de liberdade locais e globais.

Desta forma, obtém-se a matriz global  $[K]$ . Esta matriz é utilizada para determinar o vetor de deslocamentos  $\{\delta\}$ , recorrendo à seguinte expressão:

$$\{\delta\} = [K]^{-1} \cdot \{Q\} \quad (2.16)$$

Através da equação (2.16), verifica-se que é necessário calcular a inversa da matriz global  $[K]$ . No entanto, sabe-se que esta matriz apresenta uma estrutura em banda e, na forma original, o seu determinante é nulo. Consequentemente, não é possível inverter diretamente  $[K]$  sem efetuar determinados ajustes. Para que a inversão possa ser realizada,

é necessário aplicar as condições de fronteira do problema, o que implica efetuar cortes apropriados nas linhas e colunas correspondentes.

No caso de estudo em análise, as condições de fronteira fornecidas determinam quais os graus de liberdade que devem ser fixos, permitindo assim obter uma matriz  $[K]$  modificada e invertível.

As condições de fronteira da placa são definidas da seguinte forma:

- As bordas 1 e 2, correspondentes a  $x = 0$  e  $x = a$ , estão encastradas. Assim, todos os graus de liberdade são nulos nestes limites, isto é:

$$w = 0, \quad \theta_x = 0, \quad \theta_y = 0$$

- As bordas 3 e 4, correspondentes a  $y = 0$  e  $y = b$ , são livres. Consequentemente, nenhum dos parâmetros é restringido, permanecendo:

$$w \neq 0, \quad \theta_x \neq 0, \quad \theta_y \neq 0$$

O vetor de corte é construído de forma a identificar a posição dos nós pertencentes a cada elemento da placa. Dependendo da localização de cada nó, se ele se encontra numa das bordas encastradas ou nas zonas livres, os valores correspondentes a  $w$ ,  $\theta_x$  e  $\theta_y$  são anulados nas bordas encastradas ou mantêm-se diferentes de zero nas regiões livres.

Além disso, é necessário determinar o vetor de forças nodais  $\{Q\}$ . Este vetor é calculado com base na sobreposição dos nós, ou seja, dependendo se um nó se encontra isolado, partilhado por dois elementos ou por quatro elementos. Quando um nó é único, por exemplo num canto da placa, aplica-se a força  $P_1$  segundo

$$P_1 = \frac{p_0 a b}{4 n_{\text{elementos}}}. \quad (2.17)$$

Quando um nó é comum a dois elementos, por exemplo numa borda da placa, a força aplicada é  $P_2$ :

$$P_2 = \frac{p_0 a b}{2 n_{\text{elementos}}}. \quad (2.18)$$

Quando um nó é compartilhado por quatro elementos, por exemplo no centro da placa, aplica-se a força  $P_3$ :

$$P_3 = \frac{p_0 a b}{n_{\text{elementos}}}. \quad (2.19)$$

Depois de gerado o vetor de corte, aplicam-se as condições de fronteira ao vetor  $\{Q\}$ . Desta forma, este vetor passa a possuir o mesmo número de linhas que a matriz quadrada  $[K]_{\text{global}}$  reduzida, permitindo efetuar corretamente a multiplicação matricial.

Em seguida, inverte-se a matriz  $[K]_{\text{global}}$  reduzida e realiza-se a operação definida na equação (2.16): obtendo-se assim o vetor de deslocamentos  $\{\delta\}$ , que representa as deflexões em cada nó da placa.

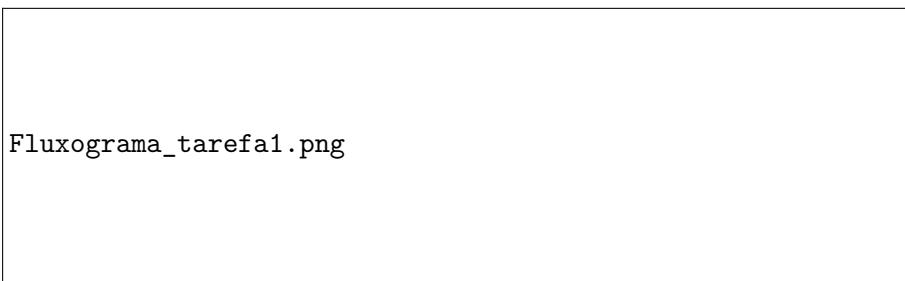
Por fim, para determinar as reações nos apoios, aplica-se a expressão derivada da equação (2.20):

$$\{R\} = [K]_{\text{global}} \cdot \{\delta\} - \{Q\}, \quad (2.20)$$

onde  $\{R\}$  representa o vetor das reações de apoio.

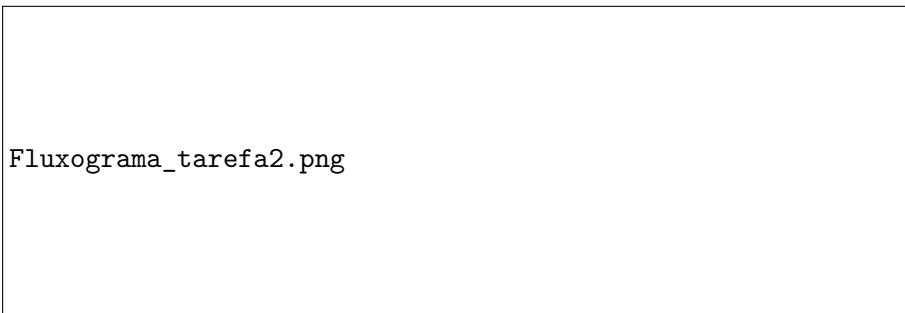
# Capítulo 3

## Implementação do código



Fluxograma\_tarefa1.png

Figura 3.1: Fluxograma referente á Tarefa 1



Fluxograma\_tarefa2.png

Figura 3.2: Fluxograma referente à Tarefa 2

Na tarefa 1, para resolver computacionalmente o problema proposto, foi utilizada a linguagem de programação *Python*. Inicialmente, são definidas as variáveis simbólicas necessárias para o desenvolvimento da formulação, nomeadamente  $x$ ,  $y$ ,  $a$ ,  $b$ ,  $c$ ,  $D$ ,  $\nu$ ,  $\omega$ ,  $\rho$  e  $t$ . Posteriormente, é efetuado o cálculo das funções  $X_i(x)$  e  $Y_j(y)$  de acordo com as expressões indicadas no formulário. Com estas funções, é então possível definir  $w(x, y)$ , que representa o modo de vibração da placa. Ao derivar e integrar a equação correspondente e reorganizá-la em função de  $\omega$ , obtém-se a frequência de vibração de forma simbólica. Finalmente, é feita a representação gráfica da função  $w(x, y)$ , permitindo assim visualizar o comportamento da placa para o modo de vibração pretendido.

A resolução da Tarefa 2 foi igualmente desenvolvida em *Python*. Começa-se por declarar as variáveis simbólicas e os parâmetros indicados no enunciado. De seguida, é feita a leitura da matriz  $[D]$ , já deduzida na alínea anterior, bem como a definição da matriz  $[H]$ , de acordo com a equação (2.13), e a construção da matriz  $[C]$ , conforme a equação (2.15). A matriz de rigidez elementar  $[K]_e$  pode ser determinada diretamente, segundo a equação (2.12). O utilizador introduz o número de subdivisões da placa, o que permite gerar uma

matriz  $[K]_e$  para cada elemento. Como estas matrizes são inicialmente simbólicas, é aplicado um algoritmo que substitui as variáveis por valores reais, diferentes para cada elemento consoante a sua posição na malha. Esse algoritmo percorre todos os elementos, atribuindo as coordenadas dos nós de acordo com a distância  $dx$  (na direção x) e  $dy$  (na direção y), obtidas pela divisão das dimensões totais da placa pelo número de elementos por lado.

Com as matrizes  $[K]_e$  devidamente atualizadas, procede-se ao seu espalhamento para formar a matriz global  $[K]_{glob}$ , utilizando uma matriz de conectividade.

Segue-se a aplicação das condições de fronteira:

- Borda 1 ( $x = 0$ ): encastrada;
- Borda 2 ( $x = a$ ): encastrada;
- Borda 3 ( $y = 0$ ): livre;
- Borda 4 ( $y = b$ ): livre.

Estas condições são impostas por meio de um vetor de controlo que identifica os nós localizados nas fronteiras e aplica as restrições correspondentes, eliminando as linhas e colunas associadas na matriz global  $[K]$  e no vetor de forças  $Q$ .

O vetor  $Q$  é construído com base na posição de cada nó da malha. Caso o nó pertença a um canto da placa, aplica-se a força definida pela equação (2.17); se se encontrar numa aresta, utiliza-se a equação (2.18); e, para os nós interiores, aplica-se a força de acordo com a equação (2.19). Após a aplicação das condições de fronteira, o vetor  $Q$  é igualmente reduzido, removendo-se as componentes correspondentes aos graus de liberdade restringidos.

Com a matriz  $[K]_{glob}$  e o vetor  $Q$  devidamente ajustados, resolve-se o sistema linear invertendo a matriz reduzida e multiplicando-a pelo vetor de forças reduzido, obtendo-se o vetor  $\delta$  das deflexões nodais. Posteriormente, representam-se graficamente as deflexões na placa, permitindo observar a deformação resultante.

Por fim, calculam-se as reações nos apoios através da aplicação da equação (2.20). Os resultados são apresentados graficamente, possibilitando uma visualização clara da distribuição das reações na estrutura e do comportamento global da placa sob as condições de contorno definidas.

# Capítulo 4

## Resultados e Discussões

Neste capítulo apresentam-se e analisam-se os resultados obtidos nas Tarefas 1 e 2. Os valores calculados e as representações gráficas permitem avaliar a precisão das abordagens adotadas e compreender o comportamento estrutural da placa ortotrópica sob análise.

### 4.1 Resultados da Tarefa 1

Na Tarefa 1, foi determinada a frequência fundamental de vibração  $\omega$  e o modo de vibração correspondente de uma placa ortotrópica, com base na formulação obtida pela aplicação do método de Ritz (Equações (2.7) e (2.11)).

A equação característica resultante da anulação do termo de variação forneceu os valores próprios do sistema, sendo o menor deles o que corresponde à frequência fundamental de vibração, dada pela equação (4.1).

$$\omega = 2\sqrt{6} \cdot \sqrt{\frac{21D_{11}b^4 + 10D_{12}a^2b^2 + 5D_{22}a^4 + 20D_{66}a^2b^2}{a^4b^4\rho t}} \quad (4.1)$$

A forma associada a esta frequência foi representada graficamente através da função  $w(x, y)$ , obtida para valores unitários de  $a$ ,  $b$  e  $c_{ij}$ . A Figura 4.1 ilustra o modo fundamental de vibração calculado.

Observa-se que as zonas de maior deflexão ocorrem no centro da placa, enquanto as

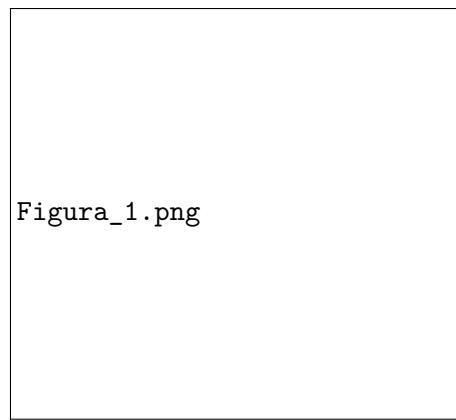


Figura 4.1: Primeiro Modo de Vibração

regiões de fronteira apresentam deslocamentos nulos, conforme as condições de fronteira impostas.

Os resultados obtidos mostram uma boa coerência com a teoria de placas finas ortotrópicas, onde o aumento do módulo de elasticidade na direção principal  $x$  ( $E_1$ ) conduz a uma rigidez superior e, consequentemente, a uma frequência natural mais elevada, tal como previsto pelas relações constitutivas da Equação (2.1).

## 4.2 Resultados da Tarefa 2

### 4.2.1 Alinea a)

A matriz de elasticidade para o material ortotrópico considerado pode ser escrita como (4.2) :

$$[D] = \frac{E_1 t^3}{12(1 - \nu_{12}\nu_{21})} \begin{bmatrix} 1 & \nu_{21} & 0 \\ \nu_{12} & \frac{E_2}{E_1} & 0 \\ 0 & 0 & \frac{G_{12}(1 - \nu_{12}\nu_{21})}{E_1} \end{bmatrix} \quad (4.2)$$

Em materiais ortotrópicos, a forma da matriz de elasticidade não coincide com a de materiais isotrópicos, pois depende das direções preferenciais do material. As propriedades  $E_1$ ,  $E_2$ ,  $\nu_{12}$ ,  $\nu_{21}$  e  $G_{12}$  traduzem o comportamento elástico ao longo dos eixos principais. Além disso, existe a relação entre os módulos nas duas direções através dos coeficientes de Poisson, podendo escrever-se  $E_2 = \frac{\nu_{12}}{\nu_{21}} E_1$  quando se considera compatibilidade entre as constantes do material. Esta representação evidencia como a rigidez flexional da placa é afetada pela anisotropia, ficando a resposta dependente da orientação das fibras e das propriedades específicas em cada direção.

### 4.2.2 Alinea b)

Após a aplicação do método descrito no capítulo anterior e a implementação do respetivo código, obtiveram-se os resultados apresentados na Figura 4.2, correspondentes às deflexões da placa para o Caso 6 considerado, utilizando uma malha composta por 411 elementos.

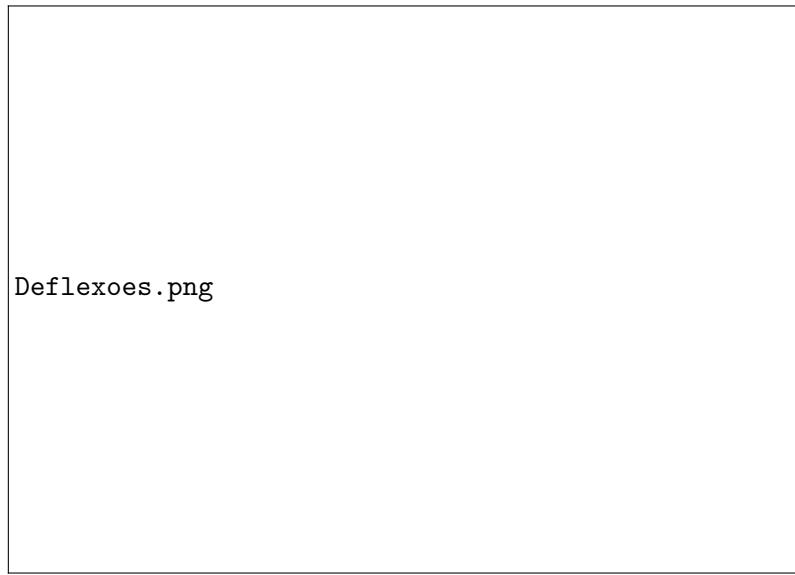
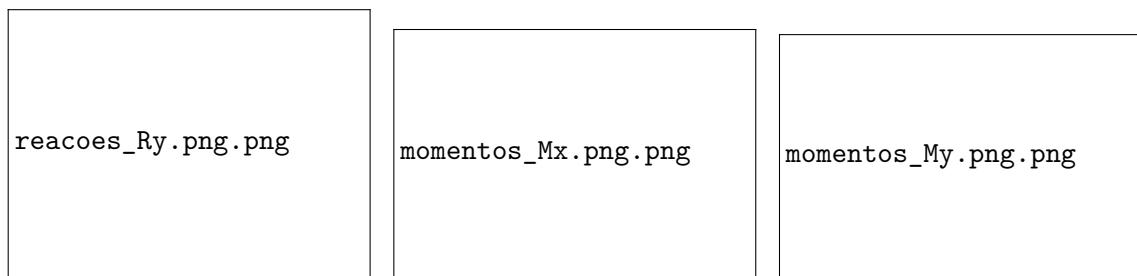


Figura 4.2: Deflexões na placa considerando uma discretização em 411 elementos.

Observa-se que, na região encastrada, as deflexões são nulas, conforme imposto pelas condições de fronteira. À medida que se progride ao longo da placa em direção à extremidade livre, o deslocamento vertical aumenta gradualmente, atingindo o valor máximo de aproximadamente  $1.72 \times 10^{-3}$  m. Este comportamento é típico de uma placa sujeita a carregamento distribuído, onde a flexão é máxima na extremidade livre e nula no encastramento.

A malha de  $21 \times 21$  elementos revelou-se adequada, assegurando um equilíbrio entre precisão e custo computacional. O refinamento adicional da discretização mostrou variações residuais nos resultados, o que confirma a convergência numérica do método.

As reações verticais e os momentos concentraram-se na zona encastrada, tal como previsto teoricamente, uma vez que é nessa região que a estrutura resiste integralmente ao carregamento aplicado. As Figuras 4.3a, 4.3b e 4.3c apresentam a distribuição das reações e momentos nos apoios.



- (a) Distribuição das reações verticais  $R_y$  junto ao encastramento.  
(b) Distribuição do momento fletor  $M_x$  junto ao encastramento.  
(c) Distribuição do momento fletor  $M_y$  junto ao encastramento.

Figura 4.3: Distribuição das reações e momentos na zona de encastramento.

Além das deflexões e das reações, foram também analisadas as tensões normais e de corte na superfície superior da placa ( $z = +t/2$ ). As Figuras 4.4, 4.5 e 4.6 apresentam, respectivamente, as distribuições das tensões  $\sigma_x$ ,  $\sigma_y$  e  $\tau_{xy}$ .

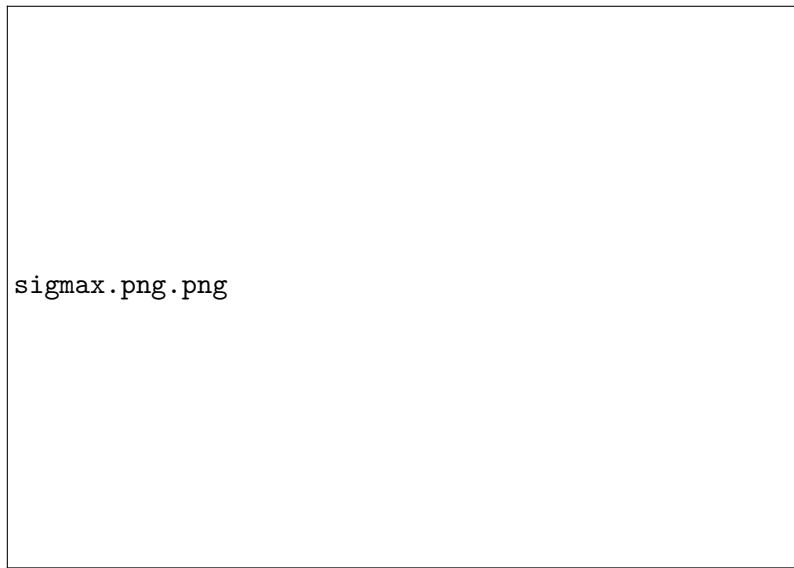


Figura 4.4: Distribuição de tensões  $\sigma_x$  na superfície superior.

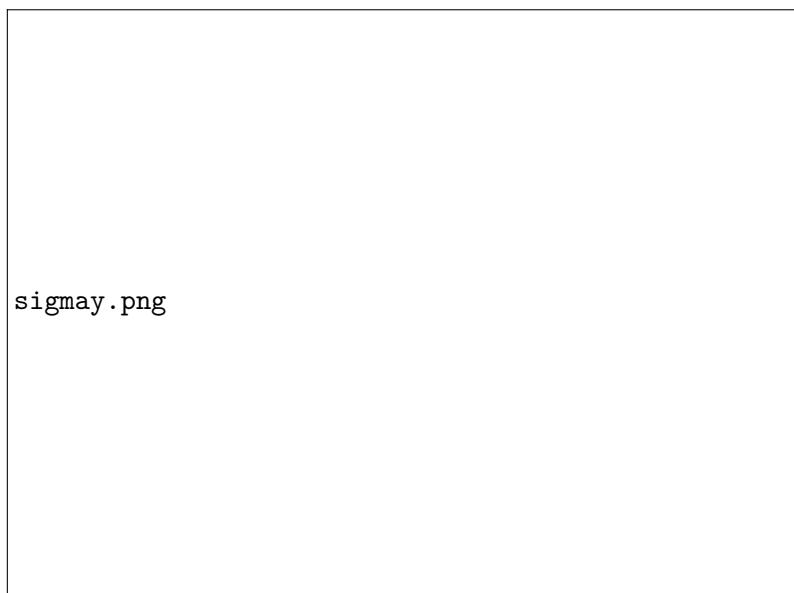


Figura 4.5: Distribuição de tensões  $\sigma_y$  na superfície superior.

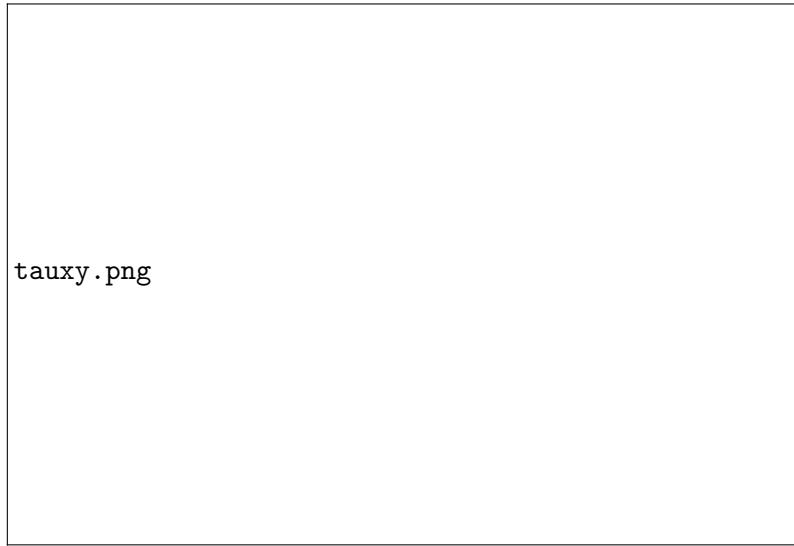


Figura 4.6: Distribuição de tensões de corte  $\tau_{xy}$  na superfície superior.

Os resultados mostram que:

- A tensão normal  $\sigma_x$  apresenta valores entre  $6.42 \times 10^6 \text{ N/m}^2$  e  $-1.18 \times 10^7 \text{ N/m}^2$ , com máximos localizados junto ao encastramento, onde ocorrem os maiores momentos fletores;
- A tensão  $\sigma_y$  varia entre  $1.23 \times 10^5 \text{ N/m}^2$  e  $-2.36 \times 10^5 \text{ N/m}^2$ , com valores significativamente inferiores, refletindo a menor influência da flexão nesta direção;
- A tensão de corte  $\tau_{xy}$  situa-se entre  $1.94 \times 10^4 \text{ N/m}^2$  e  $-2.11 \times 10^4 \text{ N/m}^2$ , atingindo os seus valores máximos próximos do encastramento, onde o gradiente de flexão é mais acentuado;
- Reação vertical  $R_y$ : entre  $8.43 \times 10^{-1} \text{ N}$  e  $1.99 \text{ N}$ ;
- Momento em torno do eixo  $x$ ,  $M_x$ : entre  $-2.42 \times 10^{-1} \text{ N}\cdot\text{m}$  e  $2.41 \times 10^{-1} \text{ N}\cdot\text{m}$ ;
- Momento em torno do eixo  $y$ ,  $M_y$ : entre  $-5.80 \times 10^{-3} \text{ N}\cdot\text{m}$  e  $6.90 \times 10^{-3} \text{ N}\cdot\text{m}$ .

Estes resultados confirmam a correta modelação das condições de fronteira, evidenciando picos de reação e momento junto à linha de encastramento, onde ocorre a total absorção dos esforços aplicados.

#### 4.2.3 Alinea c), caso (i)

Quando os eixos principais do material não coincidem com os eixos da placa, é necessário aplicar uma transformação de coordenadas. Neste cenário, as propriedades elásticas como  $E_1$ ,  $E_2$ ,  $v_{12}$ ,  $v_{21}$  e  $G_{12}$  deixam de ser diretamente associadas aos eixos  $x$  e  $y$ , exigindo a utilização de uma matriz de rotação que ajusta a orientação das fibras ao novo sistema de

referência. A matriz de rigidez transformada passa então a representar corretamente o comportamento estrutural da placa.

#### **4.2.4 Alinea c), caso (ii)**

A simetria relativamente ao plano médio simplifica parte dos cálculos, já que camadas posicionadas acima e abaixo dessa linha apresentam características semelhantes, mas com direções de fibras opostas. Esta configuração contribui para reduzir a complexidade e tornar o comportamento do compósito mais previsível.

No entanto, a ausência de balanceamento implica que as orientações das fibras não estão distribuídas de forma uniforme em torno do centro da estrutura. Esse desequilíbrio pode provocar efeitos indesejados, como torções adicionais, que alteram a distribuição das deflexões na placa.

Por fim, o aumento do número de camadas intensifica a dificuldade da análise, uma vez que cada camada deve ser considerada individualmente na construção da matriz de rigidez. É ainda necessário transformar as propriedades de cada camada para o sistema de coordenadas local, assegurando que o comportamento global da placa seja corretamente descrito.

# **Capítulo 5**

## **Conclusão**

Neste relatório apresenta um estudo numérico e computacional sobre o comportamento vibracional e estrutural de placas ortotrópicas de Kirchhoff, considerando distintas condições de fronteira e propriedades materiais. Através da aplicação de métodos computacionais de solução numérica, foi possível determinar as frequências naturais e os modos de vibração principais, verificando-se a coerência com o comportamento esperado nas zonas de encastramento e de apoio simples.

Os resultados mostraram que, ao dividir a placa em 441 elementos, se alcança um nível de precisão elevado, obtendo-se uma deflexão máxima próxima de 1.72 mm (com um erro relativo de 0.5% .

Foram igualmente analisados casos mais avançados, envolvendo materiais compósitos laminados com configurações simétricas e não balanceadas. Constatou-se que a simetria entre as camadas simplifica significativamente o processo de cálculo, enquanto o desequilíbrio entre elas pode originar efeitos indesejáveis, como a torção. Estes aspectos evidenciam a necessidade de considerar cuidadosamente a disposição e as propriedades de cada camada para uma representação rigorosa do comportamento estrutural global.

Em síntese, o trabalho desenvolvido permitiu aprofundar a compreensão dos fatores que afetam a resposta dinâmica e estrutural das placas de Kirchhoff. Ficou demonstrada a importância das características geométrica e da metodologia numérica utilizada, sendo os resultados obtidos consistentes com as previsões teóricas e validando, assim, o modelo proposto.

## **Bibliografia**

Gamboa. P.V., Apontamentos da unidade curricular – Placas e Cascas, 320 acetatos, UBI, 2020.

Gamboa, P. V., Apontamentos da Unidade Curricular – Estruturas em Materiais Compositos, 320 acetatos, Universidade da Beira Interior, Departamento de Ciências Aeroespaciais, 2025.

Dutra, T., Apontamentos da Unidade Curricular – Placas e Cascas, 250 acetatos, Universidade da Beira Interior, Departamento de Ciências Aeroespaciais, 2025.

# Apêndice A

## Implementação do código

### Tarefa 1

```
1 #Introduçao de bibliotecas
2 #Introdução dos parâmetros geométricos
3 #Calculo das rigidezes de flexão
4 #Pedir o numero de modos
5 #Definição da função X e Y
6 #Definiçao da função do deslocamento w
7 #Definição das matrizes R e B
8 #Resolver o problema generalizado de autovalores
9 #Pedir ao utilizador o modo a visualizar
10 #Reconstruir a função w(x,y)
11 #Representação simbólica da frequência omega
12 #Cálculo do deslocamento máximo |w|max
13 #Representação gráfica do modo de vibração
14
15 import sympy as sp                      # Biblioteca simbólica (derivadas e
                                             integrais)
16 import numpy as np                       # Biblioteca numérica (matrizes, vetores)
17 from scipy.linalg import eigh           # Resolver problemas generalizados de
                                             autovalores
18 import matplotlib.pyplot as plt          # Biblioteca para gráficos 3D interativos
19
20
21 a = 0.750      # [m]
22 b = 0.600      # [m]
23 t = 0.002      # [m]
24 rho = 1600     # [kg/m³]
25 E1 = 130e9     # [Pa]
26 E2 = 10e9       # [Pa]
27 nu12 = 0.26
28 G12 = 5e9      # [Pa]
29
30 nu21 = nu12 * E2 / E1
31
32 Q11 = E1 / (1 - nu12 * nu21)
33 Q22 = E2 / (1 - nu12 * nu21)
34 Q12 = nu12 * E2 / (1 - nu12 * nu21)
```

```

35 Q66 = G12
36
37 D11 = Q11 * t**3 / 12
38 D22 = Q22 * t**3 / 12
39 D12 = Q12 * t**3 / 12
40 D66 = Q66 * t**3 / 12
41
42 try:
43     n_modos = int(input("Introduz o número de modos de vibração: "))
44 except ValueError:
45     n_modos = 2
46
47 if n_modos < 1:
48     n_modos = 2
49
50 M = n_modos
51 N = n_modos
52 n = M * N    # número total de graus de liberdade
53 # =====
54 x, y = sp.symbols('x y', real=True)
55 X = []
56 Y = []
57
58 for i in range(1, M + 1):
59     X_i = (x / a)**(i + 1) - 2 * (x / a)**(i + 2) + (x / a)**(i + 3)
60     X.append(X_i)
61
62 for j in range(1, N + 1):
63     Y_j = (y / b)**j - (y / b)**(j + 1)
64     Y.append(Y_j)
65
66 c = sp.symbols(f'co:{n}', real=True)
67 w = 0
68 k = 0
69
70 for i in range(M):
71     for j in range(N):
72         w += c[k] * X[i] * Y[j]
73         k += 1
74
75 R = np.zeros((n, n), dtype=float)
76 B = np.zeros((n, n), dtype=float)
77
78 for p in range(n):
79     wp = sp.diff(w, c[p])
80     wp_xx = sp.diff(wp, x, 2)
81     wp_yy = sp.diff(wp, y, 2)

```

```

82 wp_xy = sp.diff(sp.diff(wp, x), y)
83
84 for q in range(n):
85     wq = sp.diff(w, c[q])
86     wq_xx = sp.diff(wq, x, 2)
87     wq_yy = sp.diff(wq, y, 2)
88     wq_xy = sp.diff(sp.diff(wq, x), y)
89
90     expr_R = (D11 * wp_xx * wq_xx +
91                 2 * D12 * wp_xx * wq_yy +
92                 D22 * wp_yy * wq_yy +
93                 4 * D66 * wp_xy * wq_xy)
94
95     expr_B = rho * t * wp * wq
96
97     R[p, q] = float(sp.integrate(sp.integrate(expr_R, (x, o, a)), (y, o, b))
98                     ))
99
100    B[p, q] = float(sp.integrate(sp.integrate(expr_B, (x, o, a)), (y, o, b))
101                     ))
102
103 omega2, vec = eigh(R, B)
104 omega2 = np.real(omega2[omega2 > 1e-8])
105 omega = np.sqrt(omega2)
106 freq = omega / (2 * np.pi)
107
108 print("\n==== FREQUÊNCIAS NATURAIS ===")
109 for i, f in enumerate(freq, start=1):
110     print(f"Modo {i}: f = {f:.3f} Hz")
111
112 try:
113     modo = int(input("\nIntroduz o número do modo a visualizar (ex: 1): "))
114 except ValueError:
115     modo = 1
116
117 if modo < 1 or modo > len(freq):
118     modo = 1
119
120 phi = vec[:, modo - 1]
121 phi = phi / np.max(np.abs(phi))
122
123 Nx, Ny = 80, 80
124 x_vals = np.linspace(o, a, Nx)
125 y_vals = np.linspace(o, b, Ny)
126 Xgrid, Ygrid = np.meshgrid(x_vals, y_vals)
127 W = np.zeros_like(Xgrid, dtype=float)

```

```

127 k = 0
128 for i in range(M):
129     for j in range(N):
130         X_fun = sp.lambdify(x, X[i], "numpy")
131         Y_fun = sp.lambdify(y, Y[j], "numpy")
132         W += phi[k] * X_fun(Xgrid) * Y_fun(Ygrid)
133         k += 1
134
135
136
137 x, y, c1, a_s, b_s, t_s, rho_s, D11_s, D22_s, D12_s, D66_s = sp.symbols(
138     'x y c1 a b t rho D11 D22 D12 D66', real=True)
139
140 X1 = (x / a_s)**2 - 2 * (x / a_s)**3 + (x / a_s)**4
141 Y1 = (y / b_s) - (y / b_s)**2
142 w_sym = c1 * X1 * Y1
143
144 w_xx = sp.diff(w_sym, x, 2)
145 w_yy = sp.diff(w_sym, y, 2)
146 w_xy = sp.diff(sp.diff(w_sym, x), y)
147
148 U = sp.integrate(sp.integrate(D11_s*w_xx**2 + 2*D12_s*w_xx*w_yy +
149                             D22_s*w_yy**2 + 4*D66_s*w_xy**2, (y, o, b_s)), (x,
150                                         , o, a_s))
151 T = sp.integrate(rho_s * t_s * w_sym**2, (y, o, b_s)), (x, o, a_s)
152 omega_symbolic = sp.simplify(sp.sqrt(U / T))
153
154 print("\n== EXPRESSÃO SIMBÓLICA DA FREQUÊNCIA ω ==")
155 sp.pretty_print(omega_symbolic)
156 print("\n== EXPRESSÃO EM LATEX ==")
157 print(sp.latex(omega_symbolic))
158
159 w_max = np.max(np.abs(W))
160 print(f"\nValor máximo do deslocamento relativo |w|max = {w_max:.4f}")
161
162
163
164 fig = plt.figure("Modo de vibração", figsize=(8, 6))
165 ax = fig.add_subplot(111, projection='3d')
166
167
168 ax.plot_surface(Xgrid, Ygrid, -W, cmap='turbo', edgecolor='none')
169
170 ax.set_xlabel('x (m)', color='white')
171 ax.set_ylabel('y (m)', color='white')

```

```

172 ax.set_zlabel('w(x,y)', color='white')
173 ax.set_title(f"Modo {modo} - f = {freq[modo-1]:.3f} Hz |w|max = {w_max:.4f}", color='white')
174
175
176 ax.set_facecolor('black')
177 fig.patch.set_facecolor('black')
178 ax.xaxis.label.set_color('white')
179 ax.yaxis.label.set_color('white')
180 ax.zaxis.label.set_color('white')
181 ax.tick_params(colors='white')
182 plt.show()

```

Listagem A.1: Código em Python para o caso 6

## A.1 Tarefa 2

```

1 #Importar bibliotecas
2 #Definir as propriedades do material
3 #Definir a matriz D para o caso ortotrópico
4 #Definir o tipo de borda em cada lado
5 #Definição simbolica das variaveis
6     #Definir a matriz H simbolica e inicialização em o
7     #Definir o polinomio w_e (a1...a12)
8     #Definir as rotações theta_x e theta_y (derivadas)
9     #construir a matriz simbolica C e inicialização em o
10    #Preencher a matriz C simbolica com ([1,4,7,10] -> w; [2,5,8,11] -> theta_x
11        ; [3,6,9,12] -> theta_y)
12    #Integrar simbolicamente I_xy = ∬ H' * D * H dx dy
13    #Iniciar o metodo de convergência
14    #Definir o erro relativo de convergência
15    #Iniciar a malha com 1 quadrado por lado
16    #iniciar o loop de convergência
17        #Incrementar o numero de quadrados por lado
18        #Gerar a malha de nos (cada no tem 3 graus de liberdade) e a
19            #conectividade
20        #Montar a matriz K e o vetor F e inicializa-los em o
21        # Loop pelos elementos para calcular Ke e Fe e espalhar para K, F
22        # Obter coordenadas do elemento
23        # Avaliar integral simbolica I_xy para o elemento
24        # Construir c_num
25        # Substituir em cada no e preencher as linhas correspondentes
26        # Verificar o condicionamento de C_num e calcular a inversa
27        # Calcular a matriz elementar Ke e o vetor de cargas elementar Fe
28        #Aplicar as condições de fronteira

```

```

27      #Definir a função para aplicar as condições de fronteira (seja que tipo
28      # de fronteira for)
29      #Resolver o sistema de equações lineares
30      #Montar o vetor dos deslocamentos nodais e inicializa em 0
31      #Calcular os deslocamentos nodais w
32      #Calcular o erro relativo face ao passo anterior (verificar a
33      #convergência)
34      #Usar a solução armazenada
35      #Calcular as tensões nodais
36      #Resultados numéricos e impressões finais
37      #Representação gráfica dos resultados
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

```

#Definir a função para aplicar as condições de fronteira (seja que tipo de fronteira for)

#Resolver o sistema de equações lineares

#Montar o vetor dos deslocamentos nodais e inicializa em 0

#Calcular os deslocamentos nodais w

#Calcular o erro relativo face ao passo anterior (verificar a convergência)

#Usar a solução armazenada

#Calcular as tensões nodais

#Resultados numéricos e impressões finais

#Representação gráfica dos resultados

**import** warnings # para emitir avisos controlados

**import** numpy **as** np # operações numéricas

**import** sympy **as** sp # cálculo simbólico

**import** scipy.sparse **as** sp\_sparse # matrizes esparsas

**import** scipy.sparse.linalg **as** spla # resolução de sistemas esparsos

**from** scipy.interpolate **import** griddata # interpolação para grelha (griddata)

**import** matplotlib.pyplot **as** plt # visualização / gráficos

# -----

a = 0.750 # [m]

b = 0.600 # [m]

t = 0.002 # [m]

E1 = 130e9 # [Pa]

E2 = 10e9 # [Pa]

nu12 = 0.26

G12 = 5e9 # [Pa]

po = -175.0 # [N/m^2] (negativa = para baixo)

# -----

nu21 = (E2 / E1) \* nu12

Q11 = E1 / (1 - nu12 \* nu21)

Q22 = E2 / (1 - nu12 \* nu21)

Q12 = nu21 \* Q11

Q66 = G12

Q\_mat = np.array([[Q11, Q12, 0.0],  
 [Q12, Q22, 0.0],  
 [0.0, 0.0, Q66]])

D\_b = (t\*\*3) / 12.0 \* Q\_mat

**print**("\nMatriz D\_b (flexão) [Pa·m³]:")

**print**(D\_b)

# -----

B1 = 'C' # borda x=0. encastrada

B2 = 'C' # borda x=a. encastrada

```

69 B3 = 'F'    # borda y=o. livre
70 B4 = 'F'    # borda y=b. livre
71 # -----
72 x, y, x1, x2, y1, y2 = sp.symbols('x y x1 x2 y1 y2', real=True)
73 D_sym = sp.Matrix(D_b)
74 # -----
75 H = sp.Matrix(sp.zeros(3, 12))
76 H[0, :] = sp.Matrix([0, 0, 0, -2, 0, 0, -6*x, -2*y, 0, 0, -6*x*y, 0]).T
77 H[1, :] = sp.Matrix([0, 0, 0, 0, 0, -2, 0, 0, -2*x, -6*y, 0, -6*x*y]).T
78 H[2, :] = sp.Matrix([0, 0, 0, 0, 0, -2, 0, 0, -4*x, -4*y, 0, -6*x*2, -6*y*2]).T
79 #-----
80 a_syms = sp.symbols('a1:a13', real=True)
81 w_e = (
82     a_syms[0]
83     + a_syms[1]*x
84     + a_syms[2]*y
85     + a_syms[3]*x**2
86     + a_syms[4]*x*y
87     + a_syms[5]*y**2
88     + a_syms[6]*x**3
89     + a_syms[7]*x**2*y
90     + a_syms[8]*x*y**2
91     + a_syms[9]*y**3
92     + a_syms[10]*x**3*y
93     + a_syms[11]*x*y**3
94 )
95 #-----
96 theta_x = sp.diff(w_e, x)                                # theta_x = ∂w
97                                     ∂/x
98 theta_y = sp.diff(w_e, y)                                # theta_y = ∂w
99                                     ∂/y
100 #-----
101 for row in range(12):
102     for col in range(12):
103         if row in [0, 3, 6, 9]:
104             C_sym[row, col] = sp.diff(w_e, a_syms[col])
105         elif row in [1, 4, 7, 10]:
106             C_sym[row, col] = sp.diff(theta_x, a_syms[col])
107         else:
108             C_sym[row, col] = sp.diff(theta_y, a_syms[col])
109 #-----
110 print('A integrar simbolicamente H\/*D*D*H (apenas uma vez) ... ')
111 Integrand = H.T * D_sym * H
112 I_xy = sp.integrate(sp.integrate(Integrand, (x, x1, x2)), (y, y1, y2))

```

```

113 I_xy = sp.simplify(I_xy)                                # simplificar
    expressão simbólica resultante
114 # -----
115 tol_rel_pct = 0.5                                     # tolerância
    relativa em percentagem (0.5%)
116 w_prev = np.nan                                       # valor
    anterior w_max (para comparar)
117 converged = False                                     # flag de
    convergência
118 n_side = 1                                           # inicia com 1
    quadrado por lado (malha 1x1)
119 print(f'\nIniciando sequência de refinamento automática (critério: erro <= { tol_rel_pct:.2f})\n')
120 while not converged:
121     n_side += 1                                         # incrementa
        número de quadrados por lado
122     n_quadrados = n_side * n_side                     # número total
        de quadrados
123     nnx = n_side + 1                                   # nós em x
124     nny = n_side + 1                                   # nós em y
125     n_nos = nnx * nny                               # número total
        de nós
126     dx = a / n_side                                  # Incremento em
        x
127     dy = b / n_side                                  # Incremento em
        y
128
129     coords = np.zeros((n_nos, 2))                   # inicializa
        array de coordenadas
130     idx = 0                                           # contador de
        nós
131     for j in range(0, n_side + 1):                  # varre linhas y
132         for i in range(0, n_side + 1):              # varre colunas
            x
            coords[idx, :] = [i * dx, j * dy]          # posição do nó
            (x, y)
133         idx += 1                                      # incremento do
            índice
134
135     conn = np.zeros((n_quadrados, 4), dtype=int)    # inicializa
        conectividade
136     ecount = 0                                       # contador de
        elementos
137     for j in range(1, n_side + 1):                  # varre
        elementos por linha
138         for i in range(1, n_side + 1):              # varre
            elementos por coluna

```

```

140         ecount += 1                                # índice do
141             elemento
142         n1 = (j - 1) * (n_side + 1) + i          # índice do nó
143             inferior-esquerdo (BL)
144         conn[ecount - 1, :] = [n1, n1 + 1, n1 + n_side + 2, n1 + n_side +
145             1] # BL BR TR TL
146
147 #-----
148 dofs = 3 * n_nos                            # 3 graus de
149     liberdade por nó (w, theta_x, theta_y)
150 K = sp_sparse.lil_matrix((dofs, dofs), dtype=float)
151 F = np.zeros((dofs,), dtype=float)            # inicializa
152     vetor de forças globais
153
154 #-----
155 for e in range(n_quadrados):
156     nodes = conn[e, :]                         # lista de 4 nós
157         do elemento (ainda 1-based)
158 # obter coordenadas do elemento para x e y dos 4 nós (presume-se ordem
159         BL BR TR TL)
160 Xe = coords[nodes - 1, 0]                    # x coords dos
161     nós do elemento
162 Ye = coords[nodes - 1, 1]                    # y coords dos
163     nós do elemento
164 x1v = Xe[0]                                    # x1 do elemento
165     (BL.x)
166 x2v = Xe[1]                                    # x2 do elemento
167     (BR.x)
168 y1v = Ye[0]                                    # y1 do elemento
169     (BL.y)
170 y2v = Ye[3]                                    # y2 do elemento
171     (TL.y)
172
173 I_num_mat = np.array(sp.N(I_xy.subs({x1: x1v, x2: x2v, y1: y1v, y2: y2v
174     })), dtype=float)
175 C_num = np.zeros((12, 12), dtype=float)          # inicializa
176     C_num
177 C_sub = C_sym.subs({x: x1v, y: y1v})           # valor simbólico
178     em (x1,y1) --> BL
179 C_num[0:3, :] = np.array(C_sub[0:3, :].evalf(), dtype=float)
180 C_sub = C_sym.subs({x: x2v, y: y1v})           # valor simbólico
181     em (x2,y1) --> BR
182 C_num[3:6, :] = np.array(C_sub[3:6, :].evalf(), dtype=float)
183 C_sub = C_sym.subs({x: x2v, y: y2v})           # valor simbólico
184     em (x2,y2) --> TR
185 C_num[6:9, :] = np.array(C_sub[6:9, :].evalf(), dtype=float)
186 C_sub = C_sym.subs({x: x1v, y: y2v})           # valor simbólico
187     em (x1,y2) --> TL
188 C_num[9:12, :] = np.array(C_sub[9:12, :].evalf(), dtype=float)

```

```

168
169     try:
170         cond_C = np.linalg.cond(C_num)                      # número de
171                           condição
172     except np.linalg.LinAlgError:
173         cond_C = np.inf
174     if cond_C > 1e12:
175         warnings.warn(f'C_num mal condicionado no elemento {e+1} (cond={
176             cond_C:.3e}) .')
177     try:
178         Cinv = np.linalg.inv(C_num)                      # inversa de
179                           C_num
180     except np.linalg.LinAlgError:
181         warnings.warn(f'Erro a inverter C_num no elemento {e+1}; será usado
182                         pseudo-inversa .')
183     Cinv = np.linalg.pinv(C_num)                      # matriz inversa
184                           generalizada
185
186
187     Ke = Cinv.T.dot(I_num_mat).dot(Cinv)                # montagem de Ke
188     Ae = (x2v - x1v) * (y2v - y1v)                      # área do elemento
189     Fe = np.zeros((12,), dtype=float)                    # vetor de cargas
190                           elementar
191
192     for ni in range(4):
193         Fe[ni*3 + o] = po * Ae / 4.0                   # aplicar carga po
194                           distribuída igualmente em w-dof
195
196     # espalhar Ke e Fe para K e F globais (localizar os graus de liberdade
197     # do elemento)
198     dofs_e = np.zeros((12,), dtype=int)                  # índices globais
199                           dos graus de liberdade do elemento
200
201     for iN in range(4):
202
203         node_idx = nodes[iN] - 1                        # converter 1-
204                           based -> o-based
205         dofs_e[iN*3 + o] = node_idx * 3 + o            # w DOF
206         dofs_e[iN*3 + 1] = node_idx * 3 + 1            # theta_x DOF
207         dofs_e[iN*3 + 2] = node_idx * 3 + 2            # theta_y DOF
208
209     for r in range(12):
210         for c_idx in range(12):
211             K[dofs_e[r], dofs_e[c_idx]] += Ke[r, c_idx]
212     for r in range(12):
213         F[dofs_e[r]] += Fe[r]
214
215     def apply_borda_fun(bord, coords_local, n_nos_local, dofs_local, tol_local,
216                         fixed_in):

```

```

203     fixed_out = fixed_in.copy()                      # copia do vetor
204         de fixos
205     typ = bord[ 'type' ]                            # tipo de borda
206     coord = bord[ 'coord' ]                         # coordenada
207         associada ('x' ou 'y')
208     val = bord[ 'value' ]                           # valor do borda
209         (o ou a ou b)
210     for ni in range(n_nos_local):                  # corre os nós
211         x_i = coords_local[ni, 0]                   # coordenada x do
212             nó
213         y_i = coords_local[ni, 1]                   # coordenada y do
214             nó
215         on_borda = False                           # flag se nó está
216             na borda
217         if coord == 'x':                          # se bordo
218             vertical (x fixo)
219             if abs(x_i - val) < tol_local:
220                 on_borda = True
221             else:                                # bordo horizontal
222                 (y fixo)
223                 if abs(y_i - val) < tol_local:
224                     on_borda = True
225                 if on_borda:
226                     idx_base = ni * 3                # base dos graus
227                         de liberdade do nó no vetor global
228                     if typ == 'C':                  # encastrado: w,
229                         theta_x, theta_y fixos
230                         fixed_out[idx_base + 0] = True   # fixar w
231                         fixed_out[idx_base + 1] = True   # fixar theta_x
232                         fixed_out[idx_base + 2] = True   # fixar theta_y
233                     elif typ == 'S':                # simplesmente
234                         apoiado
235                         if coord == 'y':            # borda horizontal
236                             -> w=o e theta_x=o
237                             fixed_out[idx_base + 0] = True   # w
238                             fixed_out[idx_base + 1] = True   # theta_x
239                         else:                      # borda vertical
240                             -> w=o e theta_y=o
241                             fixed_out[idx_base + 0] = True   # w
242                             fixed_out[idx_base + 2] = True   # theta_y
243                     elif typ == 'F':                # livre -> nada a
244                         fixar
245                         pass
246                     else:
247                         raise ValueError(f"Tipo de borda desconhecido: {typ} (
248                             esperado C, S ou F)")
249     return fixed_out

```

```

235
236 borda_map = {
237     'B1': { 'type': B1, 'coord': 'x', 'value': 0.0},      # B1: x=0
238     'B2': { 'type': B2, 'coord': 'x', 'value': a},        # B2: x=a
239     'B3': { 'type': B3, 'coord': 'y', 'value': 0.0},      # B3: y=0
240     'B4': { 'type': B4, 'coord': 'y', 'value': b}        # B4: y=b
241 }
242
243 tol = 1e-9                                         # tolerância para
244                                                 # identificar nós na borda
245 fixed = np.zeros((dofs,), dtype=bool)
246                                                 # aplicar cada borda ao vetor fixed
247 fixed = apply_borda_fun(borda_map['B1'], coords, n_nos, dofs, tol, fixed)
248 fixed = apply_borda_fun(borda_map['B2'], coords, n_nos, dofs, tol, fixed)
249 fixed = apply_borda_fun(borda_map['B3'], coords, n_nos, dofs, tol, fixed)
250 fixed = apply_borda_fun(borda_map['B4'], coords, n_nos, dofs, tol, fixed)
251 free = np.where(~fixed)[0]                         # índices dos DOFs
252                                                 # livres
253
254 K_csr = K.tocsr()                                 # converte para CSR
255 Kff = K_csr[free, :][:, free]                     # submatriz livre-
256                                                 # livre (esparsa)
257 Ff = F[free]                                      # vetor de forças
258                                                 # livres
259
260 try:
261     U = np.zeros((dofs,), dtype=float)               # inicializar vetor
262                                                 # de deslocamentos
263     Uf = spla.spsolve(Kff.tocsc(), Ff)            # resolve os graus de
264                                                 # liberdade livres
265     U[free] = Uf                                     # monta o vetor
266                                                 # completo U
267 except Exception as ex:
268     warnings.warn(f'Falha ao resolver sistema linear esparsa: {ex};'
269                 ' tentando solução densa.')
270     Kff_dense = Kff.toarray()
271     Uf = np.linalg.solve(Kff_dense, Ff)
272     U[free] = Uf
273
274
275 # calcular deslocamentos nodais Wnod (w DOFs, cada 3º valor começando em o)
276 Wnod = U[0::3]                                       # extrair w do vetor
277                                                 # U
278 w_max = np.max(np.abs(Wnod))                         # valor máximo
279                                                 # absoluto da deflexão nodal
280
281 # calcular erro relativo face ao passo anterior (em %)
282 if not np.isnan(w_prev):

```

```

272     erro_rel = abs(w_max - w_prev) / w_prev * 100.0
273 else:
274     erro_rel = np.inf
275
276 print(f'Iteração: malha {n_side}x{n_side} -> max|w| = {w_max:.6e} m (erro
277     rel = {erro_rel:.3f}%)')
278
279 if (not np.isnan(w_prev)) and (erro_rel <= tol_rel_pct):
280     converged = True
281     print(f'\nConvergência atingida! erro relativo = {erro_rel:.3f}\n')
282     # guardar solução convergida para pós-processamento
283     U_converged = U.copy()
284     coords_converged = coords.copy()
285     conn_converged = conn.copy()
286     n_nos_converged = n_nos
287     K_converged = K_csr.copy()
288     F_converged = F.copy()
289     Reac_converged = (K_csr.dot(U) - F)                      # reações nodais
290
291     break
292
293     w_prev = w_max
294 # -----
295 U = U_converged                                # vetor de
296     deslocamentos convergido
297 coords = coords_converged                      # coordenadas
298     convergidas
299 conn = conn_converged                          # conectividade
300     convergida
301 n_nos = n_nos_converged                       # número de nós
302     convergido
303 dofs = 3 * n_nos                             # graus de liberdade
304     totais
305 K_full = K_converged                          # matriz K
306     convergida
307 F_full = F_converged                          # vetor de forças
308     convergido
309 Reac = Reac_converged                         # vetor de reações (
310     K*U - F)
311 Wnod = U[0::3]                                 # deflexões nodais w
312
313 Reac_y = Reac[0::3]                            # forças verticais (reações de w)
314 Reac_tx = Reac[1::3]                            # momentos em torno de x (reações de theta_x)
315 Reac_ty = Reac[2::3]                            # momentos em torno de y (reações de theta_y)
316
317
318 # Identificar nós encastrados (x=0 e x=a)
319 tol = 1e-9

```

```

310 n_enc = np.where(
311     (np.abs(coords[:,o] - o.o) < tol) | (np.abs(coords[:,o] - a) < tol)
312 )[o]
313
314 # Extrair apenas as reacções nos nós encastrados
315 Ry_enc = Reac_y[n_enc]
316 Mx_enc = Reac_tx[n_enc]
317 My_enc = Reac_ty[n_enc]
318
319 print('\n--- REACÇÕES NOS ENCASTRAMENTOS ---')
320 print(f'Nº de nós encastrados: {len(n_enc)}')
321 print(f'Força vertical Ry: max = {np.max(Ry_enc):.3e} N | min = {np.min(Ry_enc):.3e} N')
322 print(f'Momento Mx: max = {np.max(Mx_enc):.3e} N·m | min = {np.min(Mx_enc):.3e} N·m')
323 print(f'Momento My: max = {np.max(My_enc):.3e} N·m | min = {np.min(My_enc):.3e} N·m')
324
325 # Usar a relação constitutiva para calcular tensões nodais
326 Qmat = Q_mat
327 sigma_x_node = np.zeros((n_nos,), dtype=float) # sigma_x por nó
328 sigma_y_node = np.zeros((n_nos,), dtype=float) # sigma_y por nó
329 tau_xy_node = np.zeros((n_nos,), dtype=float) # tau_xy por nó
330 cont = np.zeros((n_nos,), dtype=int) # contador de contribuições por nó
331 gp = np.array([-1.0/np.sqrt(3.0), 1.0/np.sqrt(3.0)])
332
333 # loop por elementos para calcular tensões nodais (média das contribuições)
334 for e in range(conn.shape[0]):
335     nodes = conn[e, :] # nós do elemento (1-based)
336     Xe = coords[nodes - 1, 0] # x coords dos 4 nós
337     Ye = coords[nodes - 1, 1] # y coords dos 4 nós
338     # deslocamentos nodais do elemento (12x1)
339     Ue = np.zeros((12,), dtype=float)
340     for iN in range(4):
341         node_idx = nodes[iN] - 1
342         Ue[iN*3:(iN*3 + 3)] = U[node_idx*3:(node_idx*3 + 3)]
343         # montar C_num novamente (como antes)
344         x1v = Xe[0]; x2v = Xe[1]; y1v = Ye[0]; y2v = Ye[3]
345         C_num = np.zeros((12, 12), dtype=float)
346         C_sub = C_sym.subs({x: x1v, y: y1v}); C_num[0:3, :] = np.array(C_sub[0:3, :].evalf(), dtype=float)
347         C_sub = C_sym.subs({x: x2v, y: y1v}); C_num[3:6, :] = np.array(C_sub[3:6, :].evalf(), dtype=float)

```

```

348 C_sub = C_sym.subs({x: x2v, y: y2v}); C_num[6:9, :] = np.array(C_sub[6:9,
349 :].evalf(), dtype=float)
350 C_sub = C_sym.subs({x: x1v, y: y2v}); C_num[9:12, :] = np.array(C_sub[9:12,
351 :].evalf(), dtype=float)
352 # inversa para obter coeficientes a_elem
353 try:
354     Cinv = np.linalg.inv(C_num)
355 except np.linalg.LinAlgError:
356     Cinv = np.linalg.pinv(C_num)
357 a_elem = Cinv.dot(Ue)                                     # coeficientes do
358                                         polinómio do elemento
359
360 # integração 2x2 Gauss para avaliar curvaturas e tensões e acumular em nós
361 for i in range(2):
362     xi = gp[i]
363     for j in range(2):
364         eta = gp[j]
365         # mapear xi, eta para coordenadas físicas x_p, y_p (assumindo
366         # mapeamento bilinear simplificado)
367         x_p = 0.5 * ((1 - xi) * x1v + (1 + xi) * x2v) # mapeamento linear
368         # em x (elementos regulares)
369         y_p = 0.5 * ((1 - eta) * y1v + (1 + eta) * y2v) # mapeamento
370         # linear em y
371         H_eval = np.array(H.subs({x: x_p, y: y_p}).evalf(), dtype=float) # avalia H no ponto (x_p, y_p)
372         kappa = H_eval.dot(a_elem)                                         # curvaturas (3x1)
373         z = t / 2.0                                                       # posição z para
374         # cálculo de tensões (face superior = +t/2)
375         sigma_vec = -z * (Qmat.dot(kappa))                                # vetor [sigma_x;
376                                         sigma_y; tau_xy] = -z * Q * kappa
377         # encontrar nó mais próximo para agregar contributo (simplificação)
378         dists = np.sum((coords - np.array([x_p, y_p]))**2, axis=1)
379         n_i = np.argmin(dists)                                              # índice do nó mais
380         # próximo (o-based)
381         sigma_x_node[n_i] += sigma_vec[0]                                    # acumular sigma_x
382         sigma_y_node[n_i] += sigma_vec[1]                                    # acumular sigma_y
383         tau_xy_node[n_i] += sigma_vec[2]                                    # acumular tau_xy
384         cont[n_i] += 1                                                       # incrementar
385             # contador de contributos
386
387 # média das contribuições em cada nó (somente onde cont>0)
388 mask = cont > 0
389 sigma_x_node[mask] = sigma_x_node[mask] / cont[mask]
390 sigma_y_node[mask] = sigma_y_node[mask] / cont[mask]
391 tau_xy_node[mask] = tau_xy_node[mask] / cont[mask]
392 #
393 print(f'--- RESULTADOS FINAIS (malha {n_side}x{n_side}) ---')

```

```

384 print(f'Deflexão máxima (|w|) = {np.max(np.abs(Wnod)):.6e} m')
385 ReacZ = Reac[0:dofs:3]                                     # reações na
   componente vertical (cada 3º)
386 print(f'Reação vertical: max = {np.max(ReacZ):.3e} N | min = {np.min(ReacZ):.3e} N')
387 if np.any(mask):
388     print('Tensões (z = +t/2) (Pa):')
389     print(f' sigma_x: max = {np.max(sigma_x_node[mask]):.3e}, min = {np.min(sigma_x_node[mask]):.3e}')
390     print(f' sigma_y: max = {np.max(sigma_y_node[mask]):.3e}, min = {np.min(sigma_y_node[mask]):.3e}')
391     print(f' tau_xy : max = {np.max(tau_xy_node[mask]):.3e}, min = {np.min(tau_xy_node[mask]):.3e}')
392 else:
393     print('Nenhuma contribuição de tensão calculada (verifica malha).')
394
395 # -----
396 Xg_lin = np.linspace(0, a, 200)                           # 200 pontos em x
397 Yg_lin = np.linspace(0, b, 200)                           # 200 pontos em y
398 Xg, Yg = np.meshgrid(Xg_lin, Yg_lin)                      # grelha 2D
399 Wgrid = griddata(points=coords, values=Wnod, xi=(Xg, Yg), method='cubic')
400
401 #Gráfico da deflexão
402 fig = plt.figure()
403 ax = fig.add_subplot(111, projection='3d')
404 surf = ax.plot_surface(Xg, Yg, Wgrid, cmap='jet', edgecolor='none')      #
   superfície
405 fig.colorbar(surf, ax=ax, shrink=0.6)                         #
   barra de cor
406 ax.set_title('Deflexão w (m) - convergido')
407 ax.set_xlabel('x (m)'); ax.set_ylabel('y (m)'); ax.set_zlabel('w (m)')
408 ax.view_init(elev=45, azim=30)
409 plt.tight_layout()
410
411 #Grafico sigma_x
412 Sxgrid = griddata(points=coords, values=sigma_x_node, xi=(Xg, Yg), method='cubic')
413 fig2 = plt.figure()
414 ax2 = fig2.add_subplot(111, projection='3d')
415 surf2 = ax2.plot_surface(Xg, Yg, Sxgrid, cmap='jet', edgecolor='none')
416 fig2.colorbar(surf2, ax=ax2, shrink=0.6)
417 ax2.set_title(r'$\sigma_x$ (Pa) - face superior'); ax2.set_xlabel('x (m)');
   ax2.set_ylabel('y (m)')
418 ax2.view_init(elev=45, azim=30)
419 plt.tight_layout()
420
421 #Grafico sigma_y

```

```

422 Sygrid = griddata(points=coords, values=sigma_y_node, xi=(Xg, Yg), method='cubic')
423 fig3 = plt.figure()
424 ax3 = fig3.add_subplot(111, projection='3d')
425 surf3 = ax3.plot_surface(Xg, Yg, Sygrid, cmap='jet', edgecolor='none')
426 fig3.colorbar(surf3, ax=ax3, shrink=0.6)
427 ax3.set_title(r'$\sigma_y$ (Pa) - face superior'); ax3.set_xlabel('x (m)'); ax3.set_ylabel('y (m)')
428 ax3.view_init(elev=45, azim=30)
429 plt.tight_layout()
430
431 #Gráfico tau_xy
432 Txygrid = griddata(points=coords, values=tau_xy_node, xi=(Xg, Yg), method='cubic')
433 fig4 = plt.figure()
434 ax4 = fig4.add_subplot(111, projection='3d')
435 surf4 = ax4.plot_surface(Xg, Yg, Txygrid, cmap='jet', edgecolor='none')
436 fig4.colorbar(surf4, ax=ax4, shrink=0.6)
437 ax4.set_title(r'$\tau_{xy}$ (Pa) - face superior'); ax4.set_xlabel('x (m)');
438 ax4.set_ylabel('y (m)')
439 ax4.view_init(elev=45, azim=30)
440 plt.tight_layout()
441
442 #Gráfico Reacções verticais
443 Rgrid = griddata(points=coords, values=ReacZ, xi=(Xg, Yg), method='cubic')
444 fig5 = plt.figure()
445 ax5 = fig5.add_subplot(111, projection='3d')
446 surf5 = ax5.plot_surface(Xg, Yg, Rgrid, cmap='jet', edgecolor='none')
447 fig5.colorbar(surf5, ax=ax5, shrink=0.6)
448 ax5.set_title('Reacções verticais (N)'); ax5.set_xlabel('x (m)'); ax5.set_ylabel('y (m)')
449 ax5.view_init(elev=45, azim=30)
450 plt.tight_layout()
451
452 #Gráficos dos momentos de reacção
453 Mxgrid = griddata(points=coords, values=Reac_tx, xi=(Xg, Yg), method='cubic')
454 Mygrid = griddata(points=coords, values=Reac_ty, xi=(Xg, Yg), method='cubic')
455
456 fig6 = plt.figure()
457 ax6 = fig6.add_subplot(111, projection='3d')
458 surf6 = ax6.plot_surface(Xg, Yg, Mxgrid, cmap='jet', edgecolor='none')
459 fig6.colorbar(surf6, ax=ax6, shrink=0.6)
460 ax6.set_title('Momento de reacção Mx (N·m)')
461 ax6.set_xlabel('x (m)'); ax6.set_ylabel('y (m)')
462 ax6.view_init(elev=45, azim=30)
463 plt.tight_layout()

```

```
464 fig7 = plt.figure()
465 ax7 = fig7.add_subplot(111, projection='3d')
466 surf7 = ax7.plot_surface(Xg, Yg, Mygrid, cmap='jet', edgecolor='none')
467 fig7.colorbar(surf7, ax=ax7, shrink=0.6)
468 ax7.set_title('Momento de reacção My (N·m)')
469 ax7.set_xlabel('x (m)'); ax7.set_ylabel('y (m)')
470 ax7.view_init(elev=45, azim=30)
471 plt.tight_layout()
472
473 plt.show()
```

Listagem A.2: Código em Python para o caso 6