MAC008 - Introdução ao Método dos Elementos Finitos - Lista 2

Antonio José de Medeiros Filho - 201965502B

1 Exercício 1

1.1 Enunciado

Seja o seguinte problema de propagação de calor transiente por condução em uma barra (Problema de valor inicial e contorno em um domínio unidimensional):

$$\rho c \frac{\partial T}{\partial t} - \kappa \frac{\partial^2 T}{\partial x^2} = 0$$

Com as seguintes condições iniciais e de contorno:

$$T(x, t = 0) = 200°C \frac{\partial T}{\partial x}(x = 0, t) = 0 T(x = L, t) = 0°C$$

A solução analítica deste problema é dada por:

$$T(x,t) = \frac{200 \cdot 4}{\pi} \cdot \sum_{n=1}^{m} \frac{(-1)^{n+1}}{2n-1} \cdot \exp\left(-\alpha \lambda_n^2 t\right) \cdot \cos\left(\lambda_n x\right)$$

Onde:

- $\lambda_n = \frac{(2n-1) \cdot \pi}{2L}$ são os autovalores da solução exata.
- $\alpha = \frac{\kappa}{\rho c}$ é o coeficiente de difusividade térmica.

Consideramos as seguintes propriedades físicas:

- $\kappa = 10 * q \frac{W}{m \cdot K}$ é o coeficiente de condutividade térmica.
- $L = 2cm$ é o comprimento do domínio.
- $\rho c = 10^6 * q \frac{J}{m^3 \cdot K}$ é a capacidade calorífica por unidade de volume do material que constitue a barra.

A variável q é dada por

$$q = 10 + \frac{N-1}{20}$$

Resolver este problema pelo método dos elementos finitos empregando interpolações lineares utilizando uma discretização espacial com uma malha de 3 elementos de mesmo comprimento. Para a solução no tempo utilize o método da diferença central usando passo de tempo o valor de $\Delta t = 1s$.

Compare gráficamente (no mesmo gráfico!) as soluções numéricas com a solução exata para os tempos $t = 80s$, $t = 100s$ e $t = 120s$.

1.2 Resolução

1.2.1 Substituindo N

Temos que $N = 1$, logo:

$$q = 10$$

1.2.2 Discretização Espacial

Vamos dividir o domínio (a barra de comprimento L) em 3 elementos de mesmo comprimento. Cada elemento terá dois nós, um em cada extremidade.

1.2.3 Formulação do Método dos Elementos Finitos

Função de Interpolação Para cada elemento, usamos funções de interpolação lineares $N_i\ (x)$ para representar a temperatura T dentro do elemento:

$$T(x) = \sum_{i=1}^{n} N_i(x) T_i$$

onde n é o número de nós por elemento e T_i é temperatura e no nó i.

Montagem da Equação de Elemento Para cada elemento, aplicamos a equação diferencial. Através da integração por partes (ou método dos resíduos ponderados), obtemos a formulação fraca do problema. Para um elemento genérico, temos:

$$\int_{x_{e-1}}^{x_e} \rho c N_i \rho \frac{\partial T}{\partial t} dx + \int_{x_{e-1}}^{x_e} \kappa \frac{\partial N_i}{\partial x} \frac{\partial T}{\partial x} dx = 0$$

Montagem da Matriz Global As equações dos elementos individuais são então montadas em uma matriz global que representa o sistema inteiro.

Aplicação das Condições de Contorno As condições de contorno são aplicadas na matriz global. A condição $T(x = L, t) = 0°C$ é uma condição de Dirichlet e é aplicada diretamente na matriz. A condição $\frac{\partial T}{\partial x}(x = 0, t) = 0$ é uma condição de Neumann e afeta a formulação da equação de elemento do nó correspondente.

1.3 Implementação

[1]:
```
# Dados do problema
N = 1
q = 10 + (N - 1) / 20
```

```
# Propriedades físicas
kappa = 10 * q  # W/(m.K)
rho_c = 10**6 * q  # J/(m^3.K)
```

[2]:
```
# Comprimento do domínio (em metros)
L = 2 / 100  # 2 cm convertido para metros

# Número de elementos e nós
num_elements = 3
num_nodes = num_elements + 1

# Comprimento de cada elemento
element_length = L / num_elements

# Posições dos nós
node_positions = [i * element_length for i in range(num_nodes)]

print("Posições dos nós:")
for i, x in enumerate(node_positions):
    print(f"  x{i} = {x:.4f} m")
```

```
Posições dos nós:
  x0 = 0.0000 m
  x1 = 0.0067 m
  x2 = 0.0133 m
  x3 = 0.0200 m
```

1.3.1 Cálculo das Matrizes Globais

As matrizes globais de massa M e rigidez K para o sistema são:

Matriz de massa M:

$$\begin{bmatrix} 22222.22 & 11111.11 & 0 & 0 \\ 11111.11 & 44444.44 & 11111.11 & 0 \\ 0 & 11111.11 & 44444.44 & 11111.11 \\ 0 & 0 & 11111.11 & 22222.22 \end{bmatrix}$$

Matriz de rigidez K:

$$\begin{bmatrix} 15000 & -15000 & 0 & 0 \\ -15000 & 30000 & -15000 & 0 \\ 0 & -15000 & 30000 & -15000 \\ 0 & 0 & -15000 & 15000 \end{bmatrix}$$

[4]:
```
import numpy as np
import sympy as sp
```

```
# Função para criar as matrizes de massa e rigidez para um elemento
def create_element_matrices(length, kappa, rho_c):
    M_element = (rho_c * length / 6) * np.array([[2, 1], [1, 2]])  # Matriz de␣
 ↪massa
    K_element = (kappa / length) * np.array([[1, -1], [-1, 1]])    # Matriz de␣
 ↪rigidez
    return M_element, K_element

# Inicialização das matrizes globais
M_global = np.zeros((num_nodes, num_nodes))
K_global = np.zeros((num_nodes, num_nodes))

# Montagem das matrizes globais
for i in range(num_elements):
    M_element, K_element = create_element_matrices(element_length, kappa, rho_c)

    # Adicionando à matriz global
    M_global[i:i+2, i:i+2] += M_element
    K_global[i:i+2, i:i+2] += K_element

print("Matriz de massa global:")
display(sp.Matrix(M_global))

print("Matriz de rigidez global:")
display(sp.Matrix(K_global))
```

Matriz de massa global:

$$\begin{bmatrix} 22222.2222222222 & 11111.1111111111 & 0 & 0 \\ 11111.1111111111 & 44444.4444444444 & 11111.1111111111 & 0 \\ 0 & 11111.1111111111 & 44444.4444444444 & 11111.1111111111 \\ 0 & 0 & 11111.1111111111 & 22222.2222222222 \end{bmatrix}$$

Matriz de rigidez global:

$$\begin{bmatrix} 15000.0 & -15000.0 & 0 & 0 \\ -15000.0 & 30000.0 & -15000.0 & 0 \\ 0 & -15000.0 & 30000.0 & -15000.0 \\ 0 & 0 & -15000.0 & 15000.0 \end{bmatrix}$$

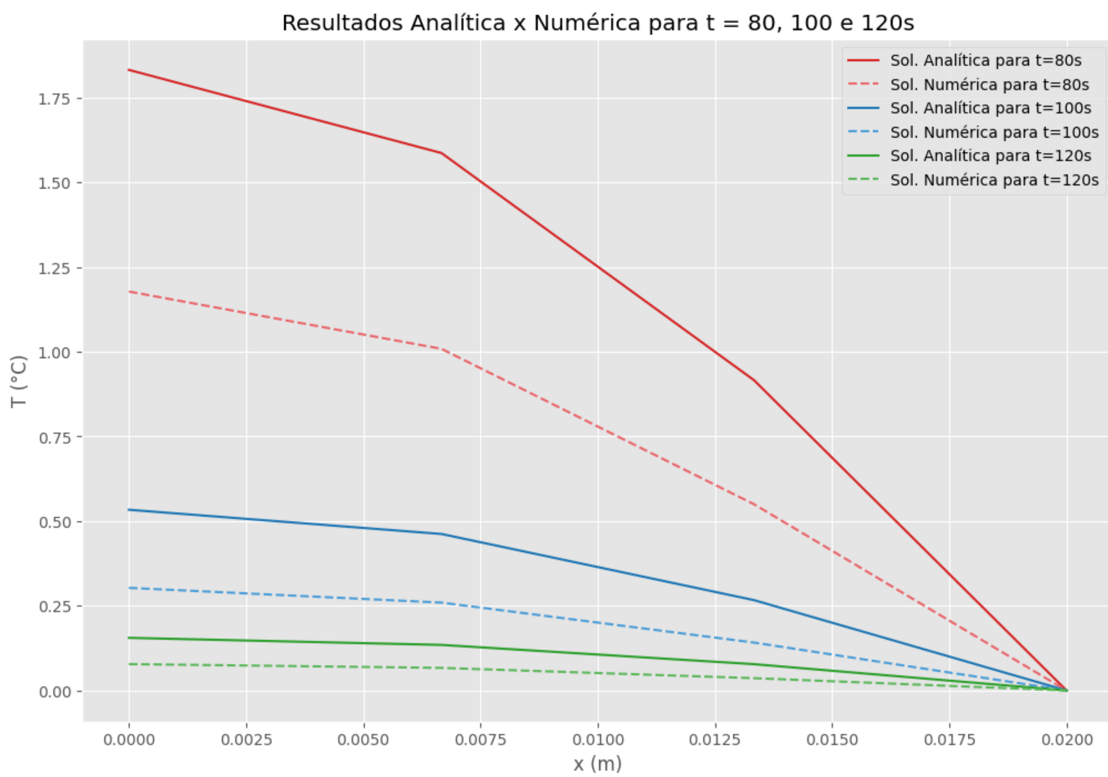
[5]:
```
# Condições iniciais
T_initial = 200  # Temperatura inicial em graus Celsius
T = np.full(num_nodes, T_initial)

# Condição de contorno: T(L, t) = 0°C
T[-1] = 0
```

```
# Passo de tempo
delta_t = 1  # segundo

# Método da diferença central: (M + delta_t/2 * K) * T_new = (M - delta_t/2 *␣
↪K) * T_old
# Como as condições de contorno não mudam com o tempo, elas são aplicadas␣
↪apenas uma vez
A = M_global + delta_t/2 * K_global
B = M_global - delta_t/2 * K_global

# Resolvendo a equação para o próximo passo de tempo
# Consideramos um único passo de tempo para este exemplo
T_new = np.linalg.solve(A, B @ T)
```

1.4 Resultados

[6]:
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

# Função para calcular a solução analítica
def analytical_solution(x, t, kappa, rho_c, L, m=3):
    alpha = kappa / rho_c
    summation = sum(((-1)**(n+1)) / (2*n - 1) * np.exp(-alpha * ((2*n - 1) * np.
↪pi / (2 * L))**2 * t) *
                    np.cos((2*n - 1) * np.pi * x / (2 * L)) for n in range(1,␣
↪m+1))
    return 200 * 4/np.pi * summation

# Tempos específicos
times = [80, 100, 120]

# Solução numérica para cada tempo
T_numerical = {time: np.full(num_nodes, T_initial) for time in times}

for time in times:
    for _ in range(int(time/delta_t)):  # Avançando no tempo
        T_numerical[time] = np.linalg.solve(A, B @ T_numerical[time])
        T_numerical[time][-1] = 0  # Aplicando a condição de contorno

# Solução analítica para cada tempo e posição
x_values = np.arange(0, L + L/3, L/3)  # Valores de x para plotagem
T_analytical = {time: [analytical_solution(x, time, kappa, rho_c, L, m=3) for x␣
↪in x_values] for time in times}

# Plotando as soluções
plt.figure(figsize=(12, 8))
```

```
shades = ['#d62728', '#ea676c', '#1f77b4', '#4c9ed9', '#2ca02c', '#5cb85c']

for time in times:
    plt.plot(x_values, T_analytical[time], color=shades.pop(0), label=f'Sol.␣
 ↪Analítica para t={time}s')
    plt.plot(node_positions, T_numerical[time], color=shades.pop(0),␣
 ↪label=f'Sol. Numérica para t={time}s', ls='--')

plt.title('Resultados Analítica x Numérica para t = 80, 100 e 120s')
plt.xlabel('x (m)')
plt.ylabel('T (°C)')
plt.legend()
plt.grid(True)
plt.show()
```



No gráfico podemos visualizar o resultado para os tempos 80s (vermelho), 100s (azul) e 120s (verde), com as soluções exatas representadas através de linhas contínuas e as soluções numéricas com linhas tracejadas.

2 Exercício 2

2.1 Enunciado

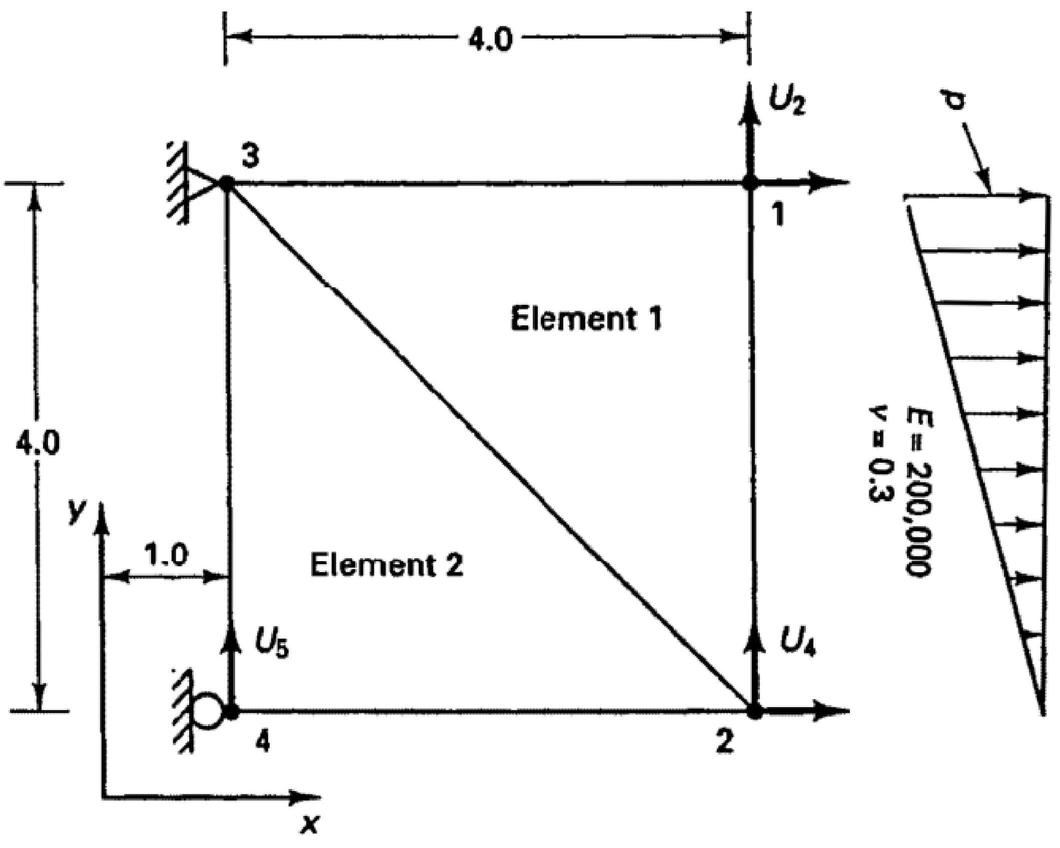
Para a viga em balanço da figura abaixo tratada como um problema de Estado Plano de Tensões determine a matriz de rigidez e o vetor de forças para as seguintes discretizações:

a) Com 2 elementos triangulares lineares conforme mostrado nesta figura.

b) Com 1 elemento retangular com os nós 1,2,3 e 4.

Determine os deslocamentos livres considerando os nós 3 e 4 presos(deslocamentos nulos).

Determine também os valores das tensões, segundo o sistema de eixos indicado nos pontos nodais e nos pontos centrais dos elementos considerados. Considere o carregamento indicado nesta figura com a distribuição de carga onde p = 150*(N+1)/20 . Adotar E (200.000) e Poisson(0.3) indicados na figura. Atente para trabalhar com unidades coerentes e que a espessura também está dada (t = 1). Utilize para as medidas do domínio o valor a = b = 4 + (N-1)/20.



2.2 Resolução

2.2.1 Substituindo N

Temos que $N = 1$, logo as dimensões do domínio serão:

$$a = b = 4$$

2.2.2 Coordenadas dos nós

- Nó 1: $(4, 4)$
- Nó 2: $(4, 0)$
- Nó 3: $(0, 4)$
- Nó 4: $(0, 0)$

2.2.3 Nós dos elementos

- Elemento 1: (1, 2, 3)
- Elemento 2: (2, 3, 4)

2.2.4 a) Elementos Triangulares

Funções de Forma Claro, vamos detalhar melhor os passos 2 e 3 para a formulação da matriz de rigidez de elementos triangulares em estado plano de tensões, focando nas funções de forma e nas relações de deformação e tensão.

Para um elemento triangular com três nós (nós 1, 2, e 3), as funções de forma lineares $ N_1 $, $ N_2 $, e $ N_3 $ podem ser definidas em termos das coordenadas dos nós. Sejam $ (x_i, y_i) $ as coordenadas dos nós $ i = 1, 2, 3 $, as funções de forma são:

$$N_i = a_i + b_i x + c_i y$$

Onde $ a_i $, $ b_i $, e $ c_i $ são constantes determinadas pela geometria do elemento e podem ser calculadas usando a fórmula:

$$a_i = \frac{x_j y_k - x_k y_j}{2A}, \quad b_i = \frac{y_j - y_k}{2A}, \quad c_i = \frac{x_k - x_j}{2A}$$

Aqui, $ (j, k) $ é um par de índices que são diferentes de $ i $, e $ A $ é a área do elemento triangular.

Deformação e Tensão A deformação em um elemento plano pode ser expressa em termos dos deslocamentos nodais. Para um elemento triangular, os deslocamentos nodais são $ u_1, v_1, u_2, v_2, u_3, v_3 $, onde $ u $ e $ v $ são os deslocamentos nos eixos $ x $ e $ y $, respectivamente. A matriz de deformação $ [B] $ relaciona a deformação com os deslocamentos nodais:

$$[B] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{bmatrix}$$

As tensões são relacionadas às deformações pela lei de Hooke para materiais isotrópicos e homogêneos. A matriz de elasticidade $ [D] $ para estado plano de tensões é:

$$[D] = \frac{E}{(1 - \nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}$$

Onde \$ E \$ é o módulo de elasticidade e \$ \$ é o coeficiente de Poisson.

A relação entre tensão e deformação é dada por:

$$[\sigma] = [D][\epsilon]$$

Onde \$ [ ] \$ é a matriz de tensão e \$ [ ] \$ é a matriz de deformação.

Combinando essas relações, podemos calcular a matriz de rigidez local para o elemento. É importante notar que, na prática, a aplicação destas fórmulas e a integração necessária para calcular a matriz de rigidez podem exigir um conhecimento detalhado de métodos numéricos, especialmente para elementos de formas mais complexas ou materiais não homogêneos.

Montagem da Matriz de Rigidez A matriz de rigidez K de um elemento triangular em estado plano de tensões é expressa por:

$$[K] = \int_A [B]^T [D][B] dA$$

$$K = tAB^T DB$$

onde:

- t é a espessura do elemento,
- A é a área do elemento triangular,
- B é a matriz de gradiente de deslocamento
- D é a matriz de elasticidade plana.

A matriz de elasticidade D é dada por:

$$D = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}$$

A matriz de deformação B para um elemento triangular é derivada das coordenadas dos nós do elemento e é dada por:

$$B = \frac{1}{2A} \begin{bmatrix} b_1 & 0 & b_2 & 0 & b_3 & 0 \\ 0 & c_1 & 0 & c_2 & 0 & c_3 \\ c_1 & b_1 & c_2 & b_2 & c_3 & b_3 \end{bmatrix}$$

onde b_i e c_i são as coordedanas do nó i do elemento.

[23]:
```
import numpy as np
import sympy as sp

# Propriedades do material
E = 200000  # Módulo de elasticidade (N/mm²)
```

```python
nu = 0.3     # Coeficiente de Poisson
t = 1        # Espessura da viga (mm)

# Dimensões da viga
a = b = 4    # Dimensões para N = 1

# Carga aplicada
p = 150 * (1 + 1) / 20

# Coordenadas dos nós (em mm)
nodes = np.array([
    [a, b],   # Nó 1
    [a, 0],   # Nó 2
    [0, b],   # Nó 3
    [0, 0],   # Nó 4
])

# Elementos (triângulos) definidos pelos nós
elements = np.array([
    [1, 2, 3],   # Elemento 1
    [2, 3, 4]    # Elemento 2
])

# Número de nós e elementos
num_nodes = len(nodes)
num_elements = len(elements)

# Função para calcular a matriz de rigidez de um elemento triangular linear
def stiffness_matrix_triangle_linear(element, nodes, E, nu, t):
    # Coordenadas dos nós do elemento
    x = nodes[element - 1, 0]
    y = nodes[element - 1, 1]

    # Área do elemento
    A = 0.5 * np.linalg.det(np.array([[1, x[0], y[0]],
                                      [1, x[1], y[1]],
                                      [1, x[2], y[2]]]))

    # Matriz B
    B = 1 / (2 * A) * np.array([
        [y[1] - y[2], 0, y[2] - y[0], 0, y[0] - y[1], 0],
        [0, x[2] - x[1], 0, x[0] - x[2], 0, x[1] - x[0]],
        [x[2] - x[1], y[1] - y[2], x[0] - x[2], y[2] - y[0], x[1] - x[0], y[0]
 ↪- y[1]]
    ])

    print('Matriz B do elemento {}:'.format(element))
```

```python
    display(sp.Matrix(B))

    # Matriz de elasticidade plana
    D = E / (1 - nu**2) * np.array([
        [1, nu, 0],
        [nu, 1, 0],
        [0, 0, (1 - nu) / 2]
    ])

    print('Matriz D do elemento {}:'.format(element))
    display(sp.Matrix(D))

    # Matriz de rigidez do elemento
    k = t * A * np.dot(np.dot(B.T, D), B)

    return k
```

[24]:
```python
k1 = stiffness_matrix_triangle_linear(elements[0], nodes, E, nu, t)
print('Matriz de Rigidez para o elemento 1:')
display(sp.Matrix(k1).evalf(3))
```

```
Matriz B do elemento [1 2 3]:
```

$$\begin{bmatrix} 0.25 & 0 & 0 & 0 & -0.25 & 0 \\ 0 & 0.25 & 0 & -0.25 & 0 & 0 \\ 0.25 & 0.25 & -0.25 & 0 & 0 & -0.25 \end{bmatrix}$$

```
Matriz D do elemento [1 2 3]:
```

$$\begin{bmatrix} 219780.21978022 & 65934.0659340659 & 0 \\ 65934.0659340659 & 219780.21978022 & 0 \\ 0 & 0 & 76923.0769230769 \end{bmatrix}$$

```
Matriz de Rigidez para o elemento 1:
```

$$\begin{bmatrix} -1.48\cdot 10^{5} & -7.14\cdot 10^{4} & 3.85\cdot 10^{4} & 3.3\cdot 10^{4} & 1.1\cdot 10^{5} & 3.85\cdot 10^{4} \\ -7.14\cdot 10^{4} & -1.48\cdot 10^{5} & 3.85\cdot 10^{4} & 1.1\cdot 10^{5} & 3.3\cdot 10^{4} & 3.85\cdot 10^{4} \\ 3.85\cdot 10^{4} & 3.85\cdot 10^{4} & -3.85\cdot 10^{4} & 0 & 0 & -3.85\cdot 10^{4} \\ 3.3\cdot 10^{4} & 1.1\cdot 10^{5} & 0 & -1.1\cdot 10^{5} & -3.3\cdot 10^{4} & 0 \\ 1.1\cdot 10^{5} & 3.3\cdot 10^{4} & 0 & -3.3\cdot 10^{4} & -1.1\cdot 10^{5} & 0 \\ 3.85\cdot 10^{4} & 3.85\cdot 10^{4} & -3.85\cdot 10^{4} & 0 & 0 & -3.85\cdot 10^{4} \end{bmatrix}$$

[25]:
```python
k2 = stiffness_matrix_triangle_linear(elements[1], nodes, E, nu, t)
print('Matriz de Rigidez para o elemento 2:')
display(sp.Matrix(k2).evalf(3))
```

```
Matriz B do elemento [2 3 4]:
```

$$\begin{bmatrix} 0.25 & 0 & 0 & 0 & -0.25 & 0 \\ 0 & 0 & 0 & 0.25 & 0 & -0.25 \\ 0 & 0.25 & 0.25 & 0 & -0.25 & -0.25 \end{bmatrix}$$

```
Matriz D do elemento [2 3 4]:
```

$$\begin{bmatrix} 219780.21978022 & 65934.0659340659 & 0 \\ 65934.0659340659 & 219780.21978022 & 0 \\ 0 & 0 & 76923.0769230769 \end{bmatrix}$$

```
Matriz de Rigidez para o elemento 2:
```

$$\begin{bmatrix} 1.1\cdot 10^5 & 0 & 0 & 3.3\cdot 10^4 & -1.1\cdot 10^5 & -3.3\cdot 10^4 \\ 0 & 3.85\cdot 10^4 & 3.85\cdot 10^4 & 0 & -3.85\cdot 10^4 & -3.85\cdot 10^4 \\ 0 & 3.85\cdot 10^4 & 3.85\cdot 10^4 & 0 & -3.85\cdot 10^4 & -3.85\cdot 10^4 \\ 3.3\cdot 10^4 & 0 & 0 & 1.1\cdot 10^5 & -3.3\cdot 10^4 & -1.1\cdot 10^5 \\ -1.1\cdot 10^5 & -3.85\cdot 10^4 & -3.85\cdot 10^4 & -3.3\cdot 10^4 & 1.48\cdot 10^5 & 7.14\cdot 10^4 \\ -3.3\cdot 10^4 & -3.85\cdot 10^4 & -3.85\cdot 10^4 & -1.1\cdot 10^5 & 7.14\cdot 10^4 & 1.48\cdot 10^5 \end{bmatrix}$$

Matriz de Rigidez Global

[11]:
```python
# Inicialização da matriz de rigidez global
K_global = np.zeros((2 * num_nodes, 2 * num_nodes))

# Montagem da matriz de rigidez global
for element in elements:
    k_local = stiffness_matrix_triangle_linear(element, nodes, E, nu, t)
    for i in range(3):
        for j in range(3):
            for k in range(2):
                for l in range(2):
                    K_global[2 * (element[i] - 1) + k, 2 * (element[j] - 1) +␣
 ↪l] += k_local[2 * i + k, 2 * j + l]

# Exibindo a matriz de rigidez global
print('Matriz de rigidez global:')
display(sp.Matrix(K_global).evalf(3))
```

```
Matriz de rigidez global:
```

$$\begin{bmatrix} -1.48\cdot 10^5 & -7.14\cdot 10^4 & 3.85\cdot 10^4 & 3.3\cdot 10^4 & 1.1\cdot 10^5 & 3.85\cdot 10^4 & 0 & 0 \\ -7.14\cdot 10^4 & -1.48\cdot 10^5 & 3.85\cdot 10^4 & 1.1\cdot 10^5 & 3.3\cdot 10^4 & 3.85\cdot 10^4 & 0 & 0 \\ 3.85\cdot 10^4 & 3.85\cdot 10^4 & 7.14\cdot 10^4 & 0 & 0 & -5.5\cdot 10^3 & -1.1\cdot 10^5 & -3.3\cdot 10^4 \\ 3.3\cdot 10^4 & 1.1\cdot 10^5 & 0 & -7.14\cdot 10^4 & 5.5\cdot 10^3 & 0 & -3.85\cdot 10^4 & -3.85\cdot 10^4 \\ 1.1\cdot 10^5 & 3.3\cdot 10^4 & 0 & 5.5\cdot 10^3 & -7.14\cdot 10^4 & 0 & -3.85\cdot 10^4 & -3.85\cdot 10^4 \\ 3.85\cdot 10^4 & 3.85\cdot 10^4 & -5.5\cdot 10^3 & 0 & 0 & 7.14\cdot 10^4 & -3.3\cdot 10^4 & -1.1\cdot 10^5 \\ 0 & 0 & -1.1\cdot 10^5 & -3.85\cdot 10^4 & -3.85\cdot 10^4 & -3.3\cdot 10^4 & 1.48\cdot 10^5 & 7.14\cdot 10^4 \\ 0 & 0 & -3.3\cdot 10^4 & -3.85\cdot 10^4 & -3.85\cdot 10^4 & -1.1\cdot 10^5 & 7.14\cdot 10^4 & 1.48\cdot 10^5 \end{bmatrix}$$

Vetor de Forças O vetor de forças para cada nó devido a uma carga distribuída é calculado integrando a carga sobre o elemento:

$$[F] = \int_A [S]^T p dA$$

onde:

$$S = \begin{bmatrix} S_i & 0 & S_j & 0 & S_k & 0 \\ 0 & S_i & 0 & S_j & 0 & S_k \end{bmatrix}$$

$S_i = (1 - \frac{x}{a})(1 - \frac{y}{b})$

$S_j = \frac{x}{a}(1 - \frac{y}{b})$

$S_k = \frac{x}{a}\frac{y}{b}$

[15]:

```
def calcula_vetor_forca(p, t):
    F = np.zeros(6)
    F[0:3:2] = p * a * b / 2
    return F * t

F_global = calcula_vetor_forca(p, t)

print('Vetor de forças externas:')
display(sp.Matrix(F_global).evalf(3))
```

```
Vetor de forças externas:
```

$$\begin{bmatrix} 120.0 \\ 0 \\ 120.0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Aplicando Condições de Contorno Conforme o enunciado, consideraremos os nós 3 e 4 como fixos.

[19]:

```
# Aplicação das condições de contorno (nós 3 e 4 fixos)
fixed_nodes = [3, 4]
free_dof = np.setdiff1d(np.arange(2 * num_nodes), np.array([2 * (node - 1) for␣
 ↪node in fixed_nodes] + [2 * node - 1 for node in fixed_nodes]))

# Redução da matriz de rigidez e do vetor de forças para os graus de liberdade␣
 ↪livres
K_reduced = K_global[np.ix_(free_dof, free_dof)]

# Redução do vetor de forças
F_reduced = F_global[free_dof]
```

Deslocamentos Resolvendo o sistema para determinar u, chegamos em:

[20]:
```
# Resolvendo o sistema para obter os deslocamentos nos graus de liberdade livres
u_free = np.linalg.solve(K_reduced, F_reduced)

# Montagem do vetor de deslocamentos completo (incluindo zeros para graus de␣
↪liberdade fixos)
u = np.zeros(2 * num_nodes)
u[free_dof] = u_free

print('Deslocamentos:')
for node in range(num_nodes):
    print('Nó {}: ({:.8f}, {:.8f})'.format(node + 1, u[2 * node], u[2 * node +␣
↪1]))
```

```
Deslocamentos:
Nó 1: (0.00156000, -0.00713143)
Nó 2: (0.00468000, -0.01025143)
Nó 3: (0.00000000, 0.00000000)
Nó 4: (0.00000000, 0.00000000)
```

Cálculo das Tensões

[27]:
```
def stress_triangle_linear(element, nodes, u, E, nu):
    # Coordenadas dos nós do elemento
    x = nodes[element - 1, 0]
    y = nodes[element - 1, 1]

    # Área do elemento
    A = 0.5 * np.linalg.det(np.array([[1, x[0], y[0]],
                                      [1, x[1], y[1]],
                                      [1, x[2], y[2]]]))

    # Matriz B
    B = 1 / (2 * A) * np.array([
        [y[1] - y[2], 0, y[2] - y[0], 0, y[0] - y[1], 0],
        [0, x[2] - x[1], 0, x[0] - x[2], 0, x[1] - x[0]],
        [x[2] - x[1], y[1] - y[2], x[0] - x[2], y[2] - y[0], x[1] - x[0], y[0]␣
↪- y[1]]
    ])

    # Matriz de elasticidade plana
    D = E / (1 - nu**2) * np.array([
        [1, nu, 0],
        [nu, 1, 0],
        [0, 0, (1 - nu) / 2]
    ])
```

```
    # Vetor de deslocamentos do elemento
    u_element = np.array([u[2 * (node - 1):2 * node] for node in element]).
 ↪flatten()

    # Cálculo das tensões
    stress = np.dot(D, np.dot(B, u_element))

    return stress

# Correção e cálculo das tensões
stresses = np.zeros((num_elements, 3))  # Inicializando o vetor de tensões

# Cálculo das tensões para cada elemento
for i, element in enumerate(elements):
    stresses[i] = stress_triangle_linear(element, nodes, u, E, nu)

print('Tensões nos elementos:')
for element in range(num_elements):
    print('Elemento {}: ({:.2f}, {:.2f}, {:.2f}) Pa'.format(element + 1,␣
 ↪stresses[element, 0], stresses[element, 1], stresses[element, 2]))
```

```
Tensões nos elementos:
Elemento 1: (137.14, 197.14, -197.14) Pa
Elemento 2: (257.14, 77.14, -197.14) Pa
```

2.2.5 b) Elemento Retangular

Funções de Forma As funções de forma para elementos retangulares são bilineares e dependem das coordenadas dos nós.

Deformação e Tensão: Para um elemento retangular em análise de elementos finitos, as funções de forma são tipicamente bilineares. Supondo um elemento retangular com quatro nós, as funções de forma podem ser expressas em termos das coordenadas locais ξ e η, que variam de -1 a 1. Para cada nó i, a função de forma $N_i(\xi, \eta)$ é definida como:

- Para o nó 1 (canto inferior esquerdo): $N_1 = \frac{1}{4}(1-\xi)(1-\eta)$
- Para o nó 2 (canto inferior direito): $N_2 = \frac{1}{4}(1+\xi)(1-\eta)$
- Para o nó 3 (canto superior direito): $N_3 = \frac{1}{4}(1+\xi)(1+\eta)$
- Para o nó 4 (canto superior esquerdo): $N_4 = \frac{1}{4}(1-\xi)(1+\eta)$

A relação entre tensão e deformação em um estado plano de tensões é dada pela lei de Hooke. Para um material isotrópico linear elástico, a matriz de elasticidade $[D]$ é usada para relacionar a tensão σ à deformação ε:

$$[D] = \frac{E}{(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}$$

Onde E é o módulo de elasticidade e ν é o coeficiente de Poisson do material. A tensão σ é então calculada como $\sigma = [D]\varepsilon$.

A relação deformação-deslocamento é expressa como $\varepsilon = [B]\{d\}$, onde $\{d\}$ é o vetor dos deslocamentos nodais.

Cálculo das Matrizes de Rigidez A formulação é similar:

$$[K] = \int_A [B]^T [D][B]\, dA$$

[30]:
```
import numpy as np

# Propriedades do material
E = 200000  # Módulo de elasticidade (N/mm²)
nu = 0.3    # Coeficiente de Poisson
t = 1       # Espessura da viga (mm)

# Dimensões da viga para N = 1
a = b = 4   # Dimensões

# Carga aplicada
p = 150 * (1 + 1) / 20

# Coordenadas dos nós (em mm)
nodes = np.array([
    [0, 0],  # Nó 1
    [a, 0],  # Nó 2
    [a, b],  # Nó 3
    [0, b]   # Nó 4
])

# Número de nós
num_nodes = len(nodes)

# Função para calcular a matriz de rigidez de um elemento retangular
def stiffness_matrix_rectangular(E, nu, t, a, b):
    # Matriz de elasticidade plana
    D = E / (1 - nu**2) * np.array([
        [1, nu, 0],
        [nu, 1, 0],
        [0, 0, (1 - nu) / 2]
    ])

    # Matriz de rigidez do elemento retangular
    k = t * a * b / 36 * np.array([
        [4, 2, -1, -2, -2, -1, -1, 0],
        [2, 4, 0, -1, -1, -2, -2, -1],
```

```
        [-1, 0, 4, 2, 1, 0, -2, -1],
        [-2, -1, 2, 4, 0, 1, -1, -2],
        [-2, -1, 1, 0, 4, 2, -1, 0],
        [-1, -2, 0, 1, 2, 4, 0, -1],
        [-1, -2, -2, -1, -1, 0, 4, 2],
        [0, -1, -1, -2, 0, -1, 2, 4]
    ])

    return k

# Aplicando a matriz de rigidez para um elemento retangular
K_rectangular = stiffness_matrix_rectangular(E, nu, t, a, b)
print('Matriz de rigidez para um elemento retangular:')
display(sp.Matrix(K_rectangular).evalf(3))
```

```
Matriz de rigidez para um elemento retangular:
```

$$\begin{bmatrix} 1.78 & 0.889 & -0.444 & -0.889 & -0.889 & -0.444 & -0.444 & 0 \\ 0.889 & 1.78 & 0 & -0.444 & -0.444 & -0.889 & -0.889 & -0.444 \\ -0.444 & 0 & 1.78 & 0.889 & 0.444 & 0 & -0.889 & -0.444 \\ -0.889 & -0.444 & 0.889 & 1.78 & 0 & 0.444 & -0.444 & -0.889 \\ -0.889 & -0.444 & 0.444 & 0 & 1.78 & 0.889 & -0.444 & 0 \\ -0.444 & -0.889 & 0 & 0.444 & 0.889 & 1.78 & 0 & -0.444 \\ -0.444 & -0.889 & -0.889 & -0.444 & -0.444 & 0 & 1.78 & 0.889 \\ 0 & -0.444 & -0.444 & -0.889 & 0 & -0.444 & 0.889 & 1.78 \end{bmatrix}$$

Deslocamentos

[31]:
```
# Aplicando as condições de contorno (nós 3 e 4 fixos)
fixed_nodes = [3, 4]
free_dof_rect = np.setdiff1d(np.arange(2 * num_nodes), np.array([2 * (node - 1)␣
 ↪for node in fixed_nodes] + [2 * node - 1 for node in fixed_nodes]))

# Redução da matriz de rigidez e do vetor de forças para os graus de liberdade␣
 ↪livres
K_reduced_rect = K_rectangular[np.ix_(free_dof_rect, free_dof_rect)]

# Vetor de forças externas (considerando a carga distribuída)
F_rect = np.zeros(2 * num_nodes)
# Aplicando a carga no nó 2 (direção y)
F_rect[2 * 2 - 1] = -p * a * b / 2  # Carga dividida igualmente entre os nós 2␣
 ↪e 3

# Redução do vetor de forças
F_reduced_rect = F_rect[free_dof_rect]

# Resolvendo o sistema para obter os deslocamentos nos graus de liberdade livres
u_free_rect = np.linalg.solve(K_reduced_rect, F_reduced_rect)
```

```python
# Montagem do vetor de deslocamentos completo (incluindo zeros para graus de␣
↪liberdade fixos)
u_rect = np.zeros(2 * num_nodes)
u_rect[free_dof_rect] = u_free_rect

# Deslocamentos nos nós para o elemento retangular
print('Deslocamentos no elemento retangular:')
for node in range(num_nodes):
    print('Nó {}: ({:.8f}, {:.8f})'.format(node + 1, u_rect[2 * node], u_rect[2␣
↪* node + 1]))
```

```
Deslocamentos no elemento retangular:
Nó 1: (-41.14285714, -7.71428571)
Nó 2: (46.28571429, -113.14285714)
Nó 3: (0.00000000, 0.00000000)
Nó 4: (0.00000000, 0.00000000)
```

Tensões

[17]:
```python
# Função para calcular as tensões em um elemento retangular
def stress_rectangular(E, nu, u, a, b):
    # Matriz de elasticidade plana
    D = E / (1 - nu**2) * np.array([
        [1, nu, 0],
        [nu, 1, 0],
        [0, 0, (1 - nu) / 2]
    ])

    # Gradientes de deslocamento
    Bx = 1 / (2 * a) * np.array([-1, 1, 1, -1])
    By = 1 / (2 * b) * np.array([-1, -1, 1, 1])

    # Deslocamentos nos nós do elemento
    u_x = u[::2]
    u_y = u[1::2]

    # Cálculo das deformações
    epsilon_x = np.dot(Bx, u_x)
    epsilon_y = np.dot(By, u_y)
    gamma_xy = np.dot(Bx, u_y) + np.dot(By, u_x)

    # Cálculo das tensões
    stress = np.dot(D, np.array([epsilon_x, epsilon_y, gamma_xy]))

    return stress

# Calculando as tensões no elemento retangular
```

```python
stress_rect = stress_rectangular(E, nu, u_rect, a, b)
print('Tensões no elemento retangular: ({:.3f}, {}, {})'.format(stress_rect[0], stress_rect[1], stress_rect[2]))
```

```
Tensões no elemento retangular: (3397959.18367347, 4040816.3265306125,
-1063186.8131868134)
```